Conception et évaluation d'un micromonde de Programmation Orientée-Objet fondé sur un jeu de construction et d'animation 3D

Fahima Djelil

N° d'ordre : D.U : 2774
EDSPIC : 783

Université Blaise Pascal - Clermont II

École Doctorale Sciences Pour L'Ingénieur de Clermont-Ferrand

Thèse

présentée par

FAHIMA DJELIL

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Spécialité : Informatique

Titre de la thèse : Conception et évaluation d'un micromonde de Programmation Orientée-Objet fondé sur un jeu de construction et d'animation 3D

Soutenue publiquement le 14 Décembre 2016 devant le jury :

M. Malik Mallem	Professeur, Université d'Évry Val d'Essonne	Rapporteur et président
M. Sébastien George	Professeur, Université du Maine	Rapporteur
Mme. Nathalie Guin	Maître de Conférences HDR, Université Claude Bernard - Lyon 1	Examinatrice
M. Eric Sanchez	Professeur, Université de Fribourg (Suisse)	Directeur
M. Jean-Marc Lavest	Professeur, Recteur, Université Française en Arménie	Directeur
M. Benjamin Albouy-Kissi	Maître de Conférences, Université d'Auvergne	Encadrant
Mme. Adélaïde Albouy-Kissi	Maître de Conférences, Université d'Auvergne	Encadrante
M. Pierre Bonton	Professeur, Université Blaise Pascal	Invité
M. Pascal Bringer	CEO Maskott	Invité

À mes amis, ma famille et à tout lecteur.

RÉSUMÉ

Les micromondes de programmation sont des environnements restreints et interactifs, dans lesquels l'apprenant apprend en interagissant avec des entités visuelles ou tangibles, sémantiquement liées à des concepts de programmation formels. Ils favorisent l'assimilation de connaissances et la compréhension de concepts abstraits de programmation au moyen de métaphores visuelles et d'expériences ludiques. Cette thèse tente d'apporter des avancées théoriques et méthodologiques sur la conception et l'évaluation de tels environnements, qui sont connus pour avoir un grand potentiel sur l'apprentissage sans que cela ne soit démontré.

Les micromondes étant des environnements d'apprentissage par le jeu, nous avons tout d'abord examiné la question du jeu et son lien à l'apprentissage. En nous appuyant sur une revue de la littérature, nous avons souligné au même titre que certains auteurs, la nécessité de distinguer le jeu-game (l'artefact informatique) du jeu-play (la situation qui découle des interactions avec le jeu-game). Le but étant de situer l'apprentissage et d'aboutir à des éléments de conception et d'évaluation de l'apprentissage. Nous nous sommes ensuite intéressés aux recherches en didactique de l'Informatique, afin d'identifier les approches d'enseignement les plus répandues visant à palier les difficultés d'apprentissage de la Programmation Orientée-Objet (POO) rencontrées par des débutants. Nous avons défini une nouvelle approche didactique pour l'introduction de la POO. Suite à cela, nous avons défini les dimensions de conception d'un micromonde, que nous désignons comme un système de représentation transitionnel, dans lequel l'apprenant développe des connaissances sur les concepts formels et abstraits de la programmation, suite à ses interactions avec l'interface du micromonde.

Les avancées théoriques et méthodologiques apportées ont été mises en œuvre dans un nouveau micromonde de POO fondé sur un jeu de construction et d'animation 3D appelé PrOgO. PrOgO implémente un système de représentation transitionnel, dans lequel les concepts fondamentaux de la POO sont représentés par des graphiques 3D visuels et interactifs. Il crée un jeu-play qui découle des interactions de l'apprenant avec son interface. Jouer avec PrOgO consiste à imaginer, créer et animer des constructions 3D significatives. PrOgO peut également être déployé dans une classe multi-dispositifs, grâce au framework Tactileo conçu à cet effet. Dans l'évaluation de l'apprentissage, nous utilisons des méthodes relevant de l'analyse de l'apprentissage, par la collecte et l'analyse de traces d'interaction pour la classification et la caractérisation des apprenants. En complément à cela, nous examinons l'état des connaissances d'apprenants, au travers de tests de vérification de connaissances. Nous tentons également d'identifier par l'analyse statistique, les actions et les comportements d'apprenants qui déterminent leur progression dans l'évaluation pré/post de l'acquisition des connaissances.

MOTS CLÉS. Micromondes de Programmation, Programmation Orientée-Objet, Apprentissage par le Jeu, PrOgO, Application 3D, Analyse de l'Apprentissage, Traces d'Interaction Numériques, Didactique de l'Informatique, Tactileo.

ABSTRACT

Programming microworlds are small and interactive environments, in which the learner learns from his interactions with visual or tangible entities having a strong semantic link with formal programming concepts. They promote knowledge assimilation and abstract programming concepts understanding by the use of visual metaphors and play. This thesis attempts to contribute to theoretical and methodological advances regarding the design and the assessment of such environments, which are known to have a great potential on learning without any evidence on that.

As microworlds are game based learning environments, we first examined the gaming issue and its relation to learning. Based on a literature review, we emphasized as some authors, the need to distinguish between the game (the computing artefact) and the play (the situation that is triggered by the interactions with the game). The purpose is to analyze learning and establish concepts that will guide the design and the evaluation of learning. Then we reviewed some research on Computer Science Education, with the view to identify some widespread teaching approaches that address beginners' difficulties in learning Object-Oriented Programming (OOP). We defined a new didactic approach for OOP introduction. We then defined the design dimensions of a microworld, we refer to as a transitional representation system, in which the learner develops knowledge on programming abstract and formal concepts, as a result to his interactions with the microworld interface.

We have implemented the theoretical and methodological advances we provided, in a new OOP microworld based on a 3D constructive and animation game called PrOgO. PrOgO implements a transitional representation system, in which basic OOP concepts are depicted with visual and interactive 3D graphics. It enables play that arises from the learner's interactions with its interface. Playing with PrOgO involves to imagining, creating and animating significant 3D constructions. PrOgO can be also deployed within a multi-device classroom through the Tactileo framework, we designed for that purpose. In the evaluation of learning, we use methods belonging to learning analytics by the collection and the analysis of digital interaction logs, with the view to classify and characterize learners. In addition to this, we examine the state of learners' knowledge through test knowledge verifications. We also attempt to examine through statistical analysis, the learners' actions and behaviours that affect their progress in pre/post evaluations of gained knowledge.

KEYWORDS. Programming Microworlds, Object-Oriented Programming, Game Based Learning, PrOgO, 3D Application, Learning Analytics, Digital Interaction Logs, Computer Science Education Research, Tactileo.

PUBLICATIONS

Quelques idées et résultats sont apparus dans les publications suivantes :

— Article de journal avec comité de lecture :

Djelil, Fahima, Adélaïde Albouy-Kissi, Benjamen Albouy-kissi, Eric Sanchez et Jean-Marc Lavest (2016). « Microworlds for learning Object-Oriented Programming : Considerations from research to practice ». In : *Joural of Interactive Learning Research* 27.3, p. 265–284.

— Articles de conférences avec actes et comité de lecture :

Djelil, Fahima (2014). « Comment évaluer un jeu d'apprentissage en contexte de formation professionnelle ». In : *Rencontres de Jeunes Chercheurs en EIAH*, p. 11–16.

Djelil, Fahima, Adélaïde Albouy-Kissi, Benjamin Albouy-Kissi, Eric Sanchez et Jean-Marc Lavest (2015). « Alice, Greenfoot et Prog&Play ou comment apprendre la programmation orientée-objet par le jeu ». In : *Environnements Informatiques pour l'Apprentissage Humain (EIAH)*, p. 420–422.

Djelil, Fahima, Benjamin Albouy-Kissi, Adélaïde Albouy-Kissi, Eric Sanchez et Jean-Marc Lavest (2015). « Towards a 3D virtual game for learning Object-Oriented Programming fundamentals and C++ language - Theoretical considerations and empirical results ». In : *Proceedings of the 7th International Conference on Computer Supported Education*, p. 289–294.

Djelil, Fahima, Eric Sanchez, Benjamin Albouy-Kissi, Jean-Marc Lavest et Adélaïde Albouy-Kissi (2014). « Towards a learning game evaluation methodology in a training context : A literature review ». In : *European Conference on Games Based Learning*. T. 2. Academic Conferences International Limited, p. 676–682.

REMERCIEMENTS

Tout d'abord, je tiens à remercier les rapporteurs de cette thèse *Malik Mallem* et *Sébastien George* ainsi que *Nathalie Guin*, *Pierre Bonton* et *Pascal Bringer* qui font partie du jury.

Ce travail de thèse a bénéficié de l'aide et du soutien de plusieurs personnes.

Ainsi, mes remerciements vont d'abord à *Adélaïde Albouy-Kissi* et *Benjamin Albouy-Kissi* qui m'ont accueillie à l'IUT du Puy en Velay, et qui m'ont offert de bonnes conditions de travail pendant trois années consécutives. Je les remercie pour m'avoir laissée travailler en autonomie et pour avoir soutenu mes propositions et contribué à leurs réalisations.

Je remercie très vivement *Eric Sanchez* qui a accepté de diriger cette thèse, sa recherche et son expertise m'ont apportée une aide précieuse. Je le remercie pour le soin qu'il a toujours donné à la relecture de mes articles, notamment à ce manuscrit de thèse, et aux nombreuses questions soulevées lors de nos échanges. Son esprit critique et sa rigueur ont permis de faire avancer mes travaux le long de trois années de thèse.

Mes remerciements vont également à *Jean-Marc Lavest* qui a accepté de diriger cette thèse, et qui malgré ses différentes fonctions de directeur puis de recteur en Arménie a été réactif à l'ensemble de mes demandes.

Je remercie tout particulièrement *Delphine Huguel* qui possède des qualités de développement remarquables. Ses efforts de développement m'ont permis de concrétiser mes propositions. Je remercie également *Sandrine Mirmand* qui a développé le premier prototype expérimental de PrOgO et qui a proposé de l'appeler ainsi.

Mes remerciements vont également à *Dominique Moncorgé*, *Manuel Grand-Brochier* et *Faouzi Jaziri* pour avoir participé à la planification des expérimentations menées à l'IUT du Puy en Velay. Je les remercie également pour le soutien et les encouragements qu'ils ont manifestés à mon égard durant cette thèse. Je remercie par la même occasion *Thibault Vitry* et *Cedric Chapat* pour leur aide technique apportée lors de ces expérimentations. Merci à *Jeanne Fine* pour son aide précieuse dans l'analyse statistique des données expérimentales recueillies et pour son soutien.

Je remercie *Isabelle Vitry* et *Isabelle Portail* qui font partie du personnel enseignant de l'IUT du Puy en Velay, ainsi que *Marie-Laure Rivet* et *Patricia Issartel* qui font partie du personnel administratif. Ces personnes m'ont apporté beaucoup de soutien et beaucoup d'encouragements.

Ce travail doit également à d'autres personnes avec qui j'ai pu échanger lors de séminaires ou de conférences. De nombreuses idées sont nées au cours de discussions très

stimulantes ou suite à des retours constructifs de relecteurs. Certains échanges avec des chercheurs à l'Institut Français de l'éducation étaient très constructifs, notamment en ce qui concerne l'évaluation des apprentissages. Les échanges que j'ai pu avoir au laboratoire IBISC avec *Samir Otmane* et *Guillaume Bouyer* ont été très intéressants. De même les échanges avec *Sébastien George* lors de mon premier comité de thèse et avec *Nathalie Guin* lors de mon deuxième comité de thèse à l'université de Lyon étaient très riches.

Je remercie mes collègues à l'UHA 4.0, où j'ai été recrutée en tant qu'ATER, *Pierre Alain-Muller*, *Lhassane Idoumghar* et *Mounir Elbaz* pour leur soutien et pour m'avoir encouragée à finir la rédaction de mon manuscrit de thèse.

Je remercie mes amis et mes proches pour leur soutien.

Enfin, ce travail a été rendu possible grâce aux contributions financières de l'IUT du Puy en Velay (Université d'Auvergne) et de la Société Maskott dans le cadre du projet Tactileo.

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

ACRONYMES

ACM Association for Computing Machinery
ACP Analyse en Composantes Principales
ANSI American National Standards Institute
API Application Programming Interface
CAF Conceptual Assessment Framework
CAH Classification Ascendante Hiérarchique
DEL Diodes ÉlectroLuminescente
ECD Evidence Centered Design
EDI Environnement de Développement Intégré
EIAH Environnements Informatiques pour l'Apprentissage Humain
IEEE IEEE Computer Society
IHM Interface Homme-Machine
IP Internet Protocol
LGPL Lesser General Public License
MIT Massachusetts Institute of Technology
OGRE Object-Oriented Graphics Rendering Engine
OO Orienté-Objet
OSC Open Sound Control
POO Programmation Orientée-Objet
RV Réalité Virtuelle
SSDP Simple Service Discovery Protocol
STIC Sciences et Technologies de l'Information et de la Communication
STR Stratégie Temps Réel
STL Standard Template Library
TCP Transmission Control Protocol
TUIO Tangible User Interface Objects
UDP User Datagram Protocol
UML Unified Modeling Language
UPnP Universal Plug and Play
UUID Universally Unique IDentifier
XML eXtensible Markup Language

INTRODUCTION

Cette partie a pour objectif d'introduire le contexte actuel et généralisé marqué par des questions liées à l'enseignement et à l'apprentissage de l'informatique, et de la programmation en particulier, à quelque niveau que ce soit et sous différentes formes, notamment par l'usage de jeux numériques. Cette partie présente notre contexte de travail, décrit notre problématique incluant notre objet d'étude et nos questions de recherche, avant de présenter la structure d'organisation de l'ensemble du manuscrit.

INTRODUCTION

Avec l'avènement du numérique et des Sciences et Technologies de l'Information et de la Communications (STICs) et l'émergence de la culture numérique, la question de l'introduction de l'informatique comme discipline, à quelques niveau que ce soit, prend de plus en plus de l'importance dans de nombreux pays dont la France et les États-Unis. Le président des États-Unis lui-même, Barack Obama, a déclaré dans une interview [1] le 30 Janvier 2016 à la Maison Blanche : "*dans la nouvelle économie, l'informatique n'est plus une compétence optionnelle. C'est une compétence fondamentale, comme la lecture, l'écriture et l'arithmétique*". Cette interview a été donnée lors du lancement de l'initiative *Computer Science for All* [2] (Informatique pour tous), qui est un programme de formation à l'Informatique qui vise à inculquer des compétence en Informatique et, notamment en programmation, à des élèves américains issus de tous les niveaux scolaires sans distinction de genre ni de milieu social.

En France, l'académie des sciences publie un rapport en Mai 2013 intitulé : "*L'enseignement de l'informatique en France. Il est urgent de ne plus attendre* " (Abiteboul et al., 2013). Ce rapport traite de la place de l'informatique dans les enseignements primaire et secondaire ainsi que la formation des enseignants, et évoque l'enseignement supérieur (universités et classes préparatoires). Ses auteurs évoquent un retard conceptuel et industriel de l'Europe et de la France en particulier, dans le domaine de l'informatique par rapport aux pays les plus dynamiques, comme les États-Unis et certains pays d'Asie. Ils affirment que ce retard est en partie lié aux carences de l'enseignement de l'informatique, qui est resté longtemps réduit à l'apprentissage des seuls usages de produits de base. Ce rapport recommande un enseignement de l'informatique généralisé, en prenant un soin particulier de rendre la discipline de l'informatique attrayante aux deux genres, quelque soit leur niveau scolaire. Les auteurs soulignent également que l'initiation à la programmation est un passage obligatoire de cet enseignement, conduisant à des activités créatrices et à l'acquisition de l'autonomie des élèves. En 2015, le président François Hollande, annonce la création de la *Grande Ecole du Numérique* [3]. Un projet qui vise à développer les apprentissages aux métiers du numérique pour favoriser l'insertion professionnelle des jeunes sans emploi ni formation. Plus qu'une nécessité, il s'agit donc de former les nouvelles générations à l'informatique afin de les préparer aux métiers de demain, et de leur donner une chance équitable pour réussir.

L'enseignement et l'apprentissage de la programmation a fait l'objet de nombreuses discussions et travaux, et comprend la conception de programmes pédagogiques à tous les niveaux, notamment le cycle universitaire. Dans la communauté anglo-saxonne, l'une des démarches officielle dans la conception de programmes pédagogiques d'Informatique au niveau universitaire, est une série de rapports de recommandations publiées

1. https://www.whitehouse.gov/blog/2016/01/30/computer-science-all
2. https://chooseyourfuture.cps.edu/computer-science-for-all/what-is-cs4all
3. https ://www.grandeecolenumerique.fr

par l'Association for Computing Machinery (ACM) et l'IEEE Computer Society (IEEE)[4]. Dans la communauté francophone, plusieurs initiatives existent visant à promouvoir l'informatique en contribuant à la médiation, la culture et à l'enseignement de l'informatique, dont la *Société Informatique de France* (SIF) qui participe à plusieurs débats de société. Le colloque Didapro-DidaSTIC qui se tient tous les trois ans, s'intéresse aux questions liées à l'enseignement et à l'apprentissage de l'informatique. Au cours du temps, plusieurs problématiques ont été abordées, comprenant notamment les nouveaux défis de la conception de *curricula* et l'intégration de nouveaux outils numériques d'aide à l'apprentissage dans ces *curricula*. Il s'agit de définir les démarches conduisant à une compréhension fine des concepts de la programmation, et de trouver les solutions permettant de la faciliter. Les travaux qui se sont intéressés à cette question soutiennent l'idée que le jeu peut amener l'apprenant à construire de façon autonome, par l'expérience et la découverte des concepts complexes et abstraits de la programmation, tout en l'engageant dans des activités en lien avec son intérêt. Les micromondes de programmation, dont le jeu est une implémentation courante, sont au cœur de ces travaux. Dès les années 1980, notamment avec l'environnement LOGO, les effets positifs des micromondes sur l'apprentissage de la programmation ont été observés (Papert, 1980). Depuis, d'autres micromondes ont été présentés comme de bons supports à l'apprentissage de la programmation impérative, tels que Scratch (Resnick et al., 2009), et la POO tels que Alice (Cooper, Dann et Pausch, 2000) et Greenfoot (Kölling, 2010b).

CONTEXTE DE TRAVAIL

Cette thèse s'inscrit dans le cadre du projet *Tactileo*, lauréat de l'appel à projet *E-Education*[5] lancé par le ministère de l'*Éducation Nationale* dans le cadre des *Investissements d'Avenir* en 2013. Ce projet s'intéresse aux modalités de conception de ressources et de mise en œuvre en classe d'outils pédagogiques mobilisant les interfaces tactiles (tables interactives tangibles, tablettes, ordinateurs, tableaux et sols interactifs . . .). Son originalité réside dans les modalités de conception de ressources et de situations d'enseignement-apprentissage réunissant des acteurs divers aussi bien privés (*Maskott, Schuch productions*) que publics (*CEA, IUT du Puy-en-Velay, IFé, IGN*).

Le projet *Tactileo* se donne pour objectif de mettre en œuvre dans la salle de classe un écosystème qui véhicule une pédagogie numérique collaborative, tirant partie de la richesse des interfaces tactiles. La classe *Tactileo* est une classe multi-dispositifs tactile, qui offre un espace ouvert et collaboratif, dans lequel les enseignants peuvent diffuser des séquences de contenus (Figure 1), et superviser des activités de groupes d'élèves.

OBJET D'ÉTUDE, PROBLÉMATIQUE ET QUESTIONS DE RECHERCHE

Notre objet d'étude porte sur le jeu pour l'apprentissage de la programmation, et plus spécifiquement sur les micromondes de programmation. Les micromondes de programmation sont des environnements restreints et interactifs, dans lesquels l'apprenant

4. http://www.acm.org/education/curricula-recommendations
5. http ://eduscol.education.fr/cid72334/les-projets-e-education-soutenus.html

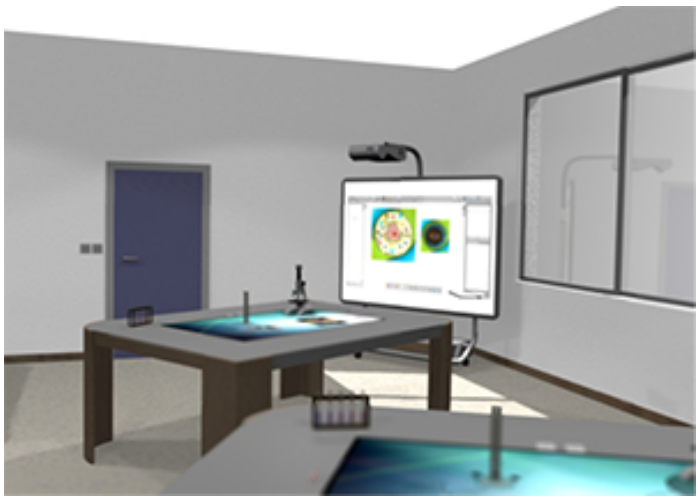


FIGURE 1 – La classe *Tactileo*.

apprend au moyen d'entités visuelles tangibles, qui sont sémantiquement liées à des concepts de programmation formels. Ils favorisent le développement et l'assimilation de connaissances abstraites par l'emploi de visualisations graphiques ou d'objets tangibles, voire l'inclusion d'éléments de jeux et l'utilisation de métaphores.

Les premiers micromondes de programmation sont apparus avec l'expansion des langages de programmation (dès la fin des années 1960). Les travaux entrepris sur les micromondes portent principalement sur l'évaluation de leur potentiel en termes d'apprentissage, sans véritablement prendre en compte la manière dont ils sont utilisés (COOPER, DANN et PAUSCH, 2003 ; DANN et al., 2012 ; XINOGALOS, SATRATZEMI et DAGDILELIS, 2006). Ainsi, leur potentiel sur l'apprentissage continue à être observé, mais n'a jamais été démontré. Plusieurs micromondes continuent à être conçus dans le but de favoriser l'apprentissage de la programmation, car leur potentiel est justement connu. Cependant très peu, voire aucune étude n'existe visant à expliquer leur potentiel et à soulever des questions permettant d'aboutir à des principes conceptuels ou à des démarches d'évaluation permettant d'examiner leur efficacité pédagogique.

Notre problématique porte sur la conception et l'évaluation d'un nouveau micromonde de POO appelé PrOgO. En ce qui concerne le problème de conception, l'obligation est de résultat, car il s'agit de répondre à la demande de la société Maskott et de l'IUT du Puy en Velay dans le cadre du projet Tactileo. Il s'agit de réaliser un artefact informatique d'aide à l'apprentissage des fondamentaux de la POO dans un contexte de jeu. En ce qui concerne le problème de l'évaluation, il s'agit de répondre à une contrainte posée par l'IFé, un des acteurs du projet Tactileo. Elle concerne la définition d'une méthodologie d'évaluation des apprentissages pour ce nouvel artefact informatique, afin de valider sa dimension pédagogique et ses apports éducatifs.

C'est dans ce contexte, que cette thèse tente d'apporter des avancées théoriques et méthodologiques sur la conception et l'évaluation de micromondes de programmation. Pour ce faire, elle tente de comprendre le jeu avant d'explorer les fondements conceptuels des micromondes de programmation qui constituent des environnements d'apprentissage par le jeu. Le but étant d'expliquer leur potentiel sur l'apprentissage de la programmation et de la POO en particulier, et de définir des principes conceptuels et

des méthodes d'évaluation, qui seront expérimentés dans un nouveau micromonde de POO.

Les micromondes constituant souvent des environnements d'apprentissage par le jeu, des questions portées sur la structure du jeu et sa relation à l'apprentissage méritent d'être posées :

1. Qu'est-ce que le jeu et quelle est sa relation à l'apprentissage ?

   Cette question possède des implications sur le processus de conception et d'évaluation des apprentissages dans un contexte de jeu, et donc par l'usage d'un micromonde :
2. Faut-il considérer la structure ludique de l'artefact informatique, ou les interactions de l'apprenant avec cet artefact ?
3. Les choix conceptuels fixés, quelle sera leur influence sur l'évaluation des apprentissages ?
4. Comment examiner l'efficacité des principes de conception déployés dans le nouvel environnement d'apprentissage conçu ?

   Une autre question qui mérite d'être posée relève de la didactique de l'informatique. Elle pose le problème de l'intégration des concepts de POO au sein de ce nouveau micromonde, et des approches visant à faciliter leur compréhension par un apprenant débutant :

5. Quelle approche didactique pour l'introduction des concepts fondamentaux de la POO peut-on implémenter au sein de ce nouvel environnement ?

   Se pose également la question des fondements conceptuels des micromondes, qui déterminent leur potentiel sur l'apprentissage de la programmation :

6. Quels sont les principes de conception des micromondes qui expliquent leur potentiel sur l'apprentissage de notions formelles de la programmation ?

   A ces questions de recherche, s'ajoute une question de contexte, qui est la classe Tactileo :

7. Comment concevoir un micromonde de façon à ce qu'il puisse s'intégrer dans la classe *Tactileo* ?

ORGANISATION DU MANUSCRIT

Ce manuscrit est organisé en deux parties essentielles. La première partie est une revue de la littérature portée sur le jeu et l'apprentissage de la programmation par le jeu, et s'organise en trois chapitres.

Le Chapitre 1 apporte des réponses aux questions (1 et 2). Il vise à décrire le jeu et son lien à l'apprentissage au travers d'une étude bibliographique. Ce chapitre donne la terminologie existante autour du jeu et de l'apprentissage par le jeu, décrit quelques modèles de conception et d'analyse de jeux à visée éducative, et présente certaines pratiques d'évaluation de jeux existantes. L'objectif étant de faire ressortir les attributs permettant de caractériser le jeu et de comprendre son lien avec l'apprentissage. Ce chapitre s'achève par une synthèse dans laquelle nous décrivons les différents attributs constitutifs d'un jeu, nous distinguons le jeu-play du jeu-game, et nous soulignons les implications de ce premier travail sur le processus de conception et d'évaluation d'un nouvel environnement d'apprentissage par le jeu. Nous présentons notre point de vue sur la conception et l'évaluation des apprentissages. Nous considérons le jeu-play comme une expérience qui peut être mise en place, par une activité dans laquelle l'apprenant est amené à exercer sa créativité et son imagination. Nous considérons également que l'évaluation des apprentissages peut se traduire par l'analyse des usages et des interactions de l'apprenant avec l'artefact conçu. De là nous répondons aux questions (3 et 4).

Le Chapitre 2 traite la question (5). Il s'intéresse à l'enseignement introductif de la programmation et évoque les difficultés rencontrées par les apprenants débutants dans leur apprentissage. Ce chapitre présente brièvement les concepts d'algorithmique et les paradigmes de programmation les plus enseignés. Il décrit également les approches d'enseignement des fondamentaux de la programmation, et de la POO en particulier en s'attardant sur l'approche *Objets-D'abord* qui a retenu l'attention de plusieurs chercheurs au sein de la communauté anglo-saxonne, et qui a été porté par l'usage de micromondes. Ce chapitre s'achève par une synthèse des différents travaux, dans laquelle nous décrivons une nouvelle approche didactique (approche par emboîtement hiérarchique pour l'introduction des fondamentaux de l'Orienté-Objet (OO)), que nous retenons dans la conception d'un nouvel environnement d'apprentissage par le jeu.

Le Chapitre 3 tend à répondre à la question (6). Il se consacre aux micromondes de programmation au travers d'interfaces et environnements de programmation tangible et de programmation visuelle existants. Le but étant, d'une part, de comprendre leur potentiel sur l'apprentissage de la programmation, et de la POO en particulier, et d'autre part, de ressortir leurs fondements conceptuels qui influencent l'assimilation et le développement des connaissances chez l'apprenant. Ce chapitre s'achève par une synthèse dans laquelle nous décrivons les nouveaux concepts que nous avons pu tirer des micromondes existants. Ces concepts concernent essentiellement notre point de vue sur les micromondes, que nous décrivons comme des systèmes de représentation transitionnels et des espaces de créativité et d'imagination, combinant l'apprentissage intuitif et le jeu-play. Nous déduirons que les micromondes ont la capacité de réduire l'écart avec les connaissances formelles de la programmation. Enfin, nous poserons un schéma général montrant les dimensions conceptuelles qui expliquent le potentiel des micromondes sur l'apprentissage de la programmation.

La seconde partie du manuscrit décrit comment l'ensemble de nos propositions découlant de notre étude bibliographique, ont été mises en œuvre dans un nouveau micromonde de POO. Cette partie s'organise en trois chapitres.

Le Chapitre 4 tend à répondre à la question (7). Il décrit un framework d'applications conçu dans le but de répondre aux exigences de la classe *Tactileo*, dans laquelle le nouveau micromonde que nous visons à concevoir doit être déployé. Ce framework fournit une interface de communication logicielle entre dispositifs hétérogènes multiples et applications Qt de haut niveau. Il offre deux interfaces d'extension indépendantes : une interface pour le gestion d'événements multi-touches et tangibles, et une interface pour la découverte automatique du réseau.

Le Chapitre 5 décrit PrOgO, un nouveau micromonde de POO dont l'objectif pédagogique est d'aider les étudiants débutants à développer des connaissances sur les concepts fondamentaux de la POO dans un contexte de jeu-play. Ce chapitre présente les fondements conceptuels de PrOgO qui découlent directement de notre étude bibliographique, à savoir, une métaphore de construction et d'animation pour la représentation de concepts fondamentaux de la POO et de systèmes OO, des visualisations graphiques 3D illustrant cette métaphore au sein d'un système de représentation transitionnel, et un jeu-play qui doit se mettre en place quand l'apprenant accepte d'interagir avec PrOgO. Ce chapitre montre également comment l'approche par emboîtement hiérarchique des fondamentaux de l'OO a été implémentée au sein de PrOgO. Il donne l'architecture conceptuelle et techniques de PrOgO, et explique les techniques d'interaction 3D implémentées. Enfin, il s'achève avec les principales forces et limites de ce nouveau micromonde.

Le Chapitre 6 se consacre à l'évaluation des usages de PrOgO et au diagnostic des connaissances d'apprenants. Il décrit trois expérimentations différentes menées entre Octobre 2014 et Mai 2016 avec trois prototypes différents de PrOgO. Il donne pour chaque étude expérimentale, ses objectifs, la méthodologie employée dans la collecte et l'analyse des données, les résultats d'analyse et les conclusions tirées de leurs discussions. Ce chapitre montre l'intérêt des techniques d'analyse de l'apprentissage dans l'analyse des comportements d'apprenants au travers de traces d'interaction numériques. Il souligne également les limites de notre méthodologie de collecte fondée sur le recueil de traces numériques d'interaction, et évoque les techniques de suivi du regard pour l'analyse des interactions utilisateurs (eye-tracking).

Enfin, ce manuscrit conclut par un bilan de nos contributions, nos perspectives de recherche et notre positionnement théorique.

Première partie

JEUX, PROGRAMMATION ET APPRENTISSAGE

Cette partie est une revue de la littérature sur l'apprentissage par le jeu, l'apprentissage et l'enseignement de la programmation et les environnements d'initiation à la programmation par le jeu. Un premier objectif consiste à comprendre le jeu et sa relation à l'apprentissage, au travers d'une revue de la littérature portée sur les modèles de conception et d'analyse de jeux conçus à des fins éducatives (Chapitre 1). Un deuxième objectif est de fournir un état des lieux des approches d'enseignement de la programmation au niveau premier cycle universitaire (Chapitre 2). Il s'agit en particulier, de connaître les difficultés d'apprentissage du paradigme OO, puis d'examiner les tendances de recherche visant à améliorer l'enseignement et l'apprentissage des fondamentaux de la POO. Enfin, un troisième objectif vise à explorer les interfaces et les environnements d'initiation à la programmation par le jeu, à ressortir leurs caractéristiques et à comprendre leur potentiel dans l'apprentissage de la programmation (Chapitre 3).

1

APPRENTISSAGE PAR LE JEU

1.1 INTRODUCTION

Le jeu est largement connu pour avoir un grand potentiel dans le développement des connaissances scientifiques chez l'individu (Kafai, 2006 ; Sanchez, 2014). Faire jouer constitue une méthode pédagogique pour laquelle il existe un intérêt dans les systèmes éducatifs. Sanchez (2014) souligne que le jeu apparaît comme une *pédagogie alternative* qui permet de prendre en compte la culture numérique des apprenants.

Certains auteurs tentent de définir le jeu et d'étudier sa relation avec l'homme (Caillois, 1991 ; Henriot, 1969 ; Huizinga, 1951) et d'autres recherches s'intéressent au lien existant entre le jeu et l'apprentissage (Sanchez, Emin-Martinez et Mandran, 2015 ; Shaffer, 2006). Il s'agit alors d'études traitant de la conception et de l'exploitation du jeu à des fins d'apprentissage, donnant lieu à une très riche littérature, incluant des principes de conception bien établis et des modèles théoriques et expérimentaux pour la conception et l'analyse de jeux dédiés à l'apprentissage (Anneta, Lamb et Stone, 2011 ; De Freitas et Oliver, 2006 ; Garris, Ahlers et Driskell, 2002 ; Kiili, 2005).

Ce chapitre vise à définir le jeu et son lien avec l'apprentissage. Pour ce faire, nous nous appuyons sur une revue de la littérature pour étayer, dans un premier temps, la terminologie existante autour du jeu et de l'apprentissage par le jeu (Section 1.2), puis dans un second temps, quelques modèles de conception et d'analyse de jeux à visée éducative (Section 1.3), ainsi que des pratiques d'évaluation de jeux existantes (Section 1.4), afin de faire ressortir les attributs permettant de caractériser le jeu et de comprendre son lien avec l'apprentissage (Section 1.5). Enfin, nous concluons par les implications de ce premier chapitre sur le déroulement de la suite de ce travail de thèse (Section 1.6).

1.2 JEU ET APPRENTISSAGE PAR LE JEU : DÉFINITIONS ET TERMINOLOGIE

Le jeu a suscité le questionnement de plusieurs auteurs (Caillois, 1991 ; Henriot, 1969 ; Huizinga, 1951 ; Sutton-Smith, 2009) et a fait l'objet de nombreuses recherches (Garris, Ahlers et Driskell, 2002 ; Kiili, 2005 ; Sanchez, Emin-Martinez et Mandran, 2015 ; Von Wangenheim, Thiry et Kochanski, 2009). Les jeux sont divers et le terme *jeu* lui-même, par des usages multiples, peut avoir des significations très similaires mais aussi très variées. En effet, certains auteurs soulignent même son caractère ambigu (Sutton-Smith, 2009). L'une des définitions du jeu les plus reprise dans la littérature, qui est aussi parfois contestée, est celle de Huizinga (1951) décrivant le jeu comme *une action ou une activité volontaire, accomplie dans certaines limites fixées de temps et de lieu, suivant une règle librement consentie mais complètement impérieuse, pourvue d'une fin en soi, accompagnée d'un sentiment de tension et de joie et d'une conscience d'être autrement que dans la vie courante*. Cette définition est contestée par Caillois (1991), qui décrit le

jeu en considérant ses différentes catégories en proposant une classification des jeux. Selon cet auteur, tous les jeux ne répondent pas au même besoin et ne traduisent pas la même attitude chez le joueur. Il définit le jeu comme une activité qui est : *libre (à laquelle le joueur ne saurait être obligé sans que le jeu perde aussitôt sa nature de divertissement), séparée (circonscrite dans des limites de temps et d'espace fixées à l'avance), incertaine (dont le déroulement ne saurait être déterminé ni le résultat acquis préalablement), improductive (ne créant ni biens, ni richesses, ni aucun élément nouveau), réglée (soumise à des conventions qui suspendent les lois ordinaires et qui instaurent momentanément une législation nouvelle) et fictive (accompagnée d'une conscience spécifique de réalité seconde ou de franche irréalité par rapport à la vie courante)*. Il propose quatre attitudes ludiques distinctes donnant lieu à quatre catégories de jeux, selon que le rôle prédominant du joueur soit celui de la compétition (*agôn*), du hasard (*alea*), du simulacre (*mimicry*) ou du vertige (*ilinx*).

Les auteurs qui tentent de définir le jeu, le font en distinguant le "*game*" qui est un jeu structuré autour de règles et le "*play*", jeu qui se déploie librement chez le joueur (Winnicott, 1971). Cette distinction a été déjà donnée par Henriot (1969) en centrant l'analyse sur *le jouer, le jouet et le joueur*. Il déclare : *qu'il s'agisse de jeu ou de jouet, ces réalités n'ont de sens et de fonction que parce qu'elles sont l'objet d'un jouer (play) qui tient lui-même au jeu que le joueur par son attitude, introduit et maintient entre son jeu et lui. Jouant à (s'il s'agit d'un jeu), avec (s'il s'agit d'un jouet), il se tient à distance. Le jeu, le jouet peuvent être définis de façon générale comme étant ce qui se prête au jeu. S'il y a jeu, le jeu n'est que dans l'attitude de l'acteur à l'égard de son acte* (Henriot, 1969). Cette distinction est également reprise par Sanchez (2014) qui utilise les termes *jeu-game* pour désigner le jeu en tant que *artefact* et *jeu-play* pour désigner la *situation* qui relève du jeu. Il donne pour exemple, l'expression "*jeu vidéo*", qui ne renvoie pas à la situation qui se met en place, mais à l'artefact utilisé pour jouer. Il précise que *game* et *play* ne s'opposent pas nécessairement puisqu'une combinaison de ces deux mots permet, en anglais, de désigner la jouabilité d'un jeu vidéo (*game-play*) en soulignant une tension entre la contrainte des règles imposées par l'architecture du jeu et la liberté offerte au joueur. Il distingue également le jeu-game du jouet : *un jouet est un artefact qui n'intègre aucune contrainte sur la manière de jouer, aucune règle qu'il faudra suivre, aucune perspective de récompense, aucun enjeu. C'est au joueur de définir les règles qui permettront l'amusement. A contrario, un jeu-game ne se limite pas à reproduire un univers mais constitue également une structure qui oriente le déroulement du jeu-play* (Sanchez, 2014).

La notion de jeu est d'autant plus complexe quand elle est mêlée à celle de l'apprentissage, donnant lieu à une large terminologie. *Jeux sérieux* est l'une des expressions la plus utilisée pour désigner des jeux vidéo employés à des fins utilitaires, en particulier d'éducation, de formation ou d'entrainement, et c'est le plus souvent l'expression anglaise "*serious game*" qui est employée (Sanchez, 2014). Le serious game est défini comme une *application informatique, dont l'objectif est de combiner à la fois des aspects sérieux (serious) tels, de manière non exhaustive, l'enseignement, l'apprentissage, la communication, ou encore l'information, avec des ressorts ludiques issus du jeu vidéo (game). Une telle association, qui s'opère par l'implémentation d'un "scénario pédagogique", a donc pour but de s'écarter du simple divertissement* (Alvarez, 2007).

Quand le jeu est développé pour des visées éducatives, certains auteurs utilisent le terme "*ludo-éducatif*", équivalent au terme anglais "*edutainment*", pour parler du fait de

présenter sous forme de jeu vidéo un contenu éducatif, en insérant des séquences ludiques avec des défis et des récompenses (Natkin, 2009). De là découle l'expression "*jeu vidéo éducatif*" pour désigner *un jeu vidéo dont les mécanismes d'immersion et d'apprentissage sont exploités pour améliorer certaines compétences et connaissances du joueur* (Natkin, 2009). Cette expression est souvent déclinée en "*jeu éducatif*", en anglais "*educational game*" (Moreno-Ger et al., 2008), "*jeu pédagogique*", en anglais "*pedagogical game*" (Sindre, Natvig et Jahre, 2009) ou "*instructional game*" (Klein et Freitag, 1991), ou "*jeu d'apprentissage*", en anglais "*learning game*" (Barber, 1997 ; Marfisi-Schottman, 2012).

Il s'agit aussi pour certains auteurs de trouver l'expression la plus précise pour désigner un jeu en soulignant qu'il a été conçu avec des visées éducatives. C'est pour cela que Shaffer (2006) emploie l'expression "*jeu numérique épistémique*" (*digital epistemic game*) pour désigner *des situations d'apprentissage ludiques élaborées à l'aide de technologies numériques*. Cette expression est reprise par Sanchez (2014) qui affirme que, dans ce contexte, le terme "*épistémique*" est plus précis que le terme "*sérieux*", car il est utilisé pour qualifier *les interactions qui jouent un rôle dans la construction des connaissances scientifiques chez l'apprenant*. Il explique également que l'emploi du terme "*situation*" [1] plutôt que "*artefact*" est un moyen de considérer le jeu et son rôle dans le processus d'apprentissage. Il voit que l'expression jeu sérieux est problématique, en expliquant que l'interactivité n'est pas intrinsèque à l'artefact informatique, mais elle émerge d'un dispositif *socio-technique* (Sanchez, 2014). L'utilisation de l'artefact induit la mise en place d'une situation dans laquelle se développent des interactions, dites "*épistémiques*" quand l'apprenant accepte d'interagir. De ces interactions émerge un sentiment de plaisir lié au fait que l'expérience vécue présente des caractéristiques ludiques (Sanchez, 2014).

La "*ludicisation*" est un autre terme émergent, utilisé pour désigner des situations d'apprentissage comportant une jouabilité et permettant à l'individu d'adopter une attitude ludique, en utilisant des dispositifs qui ne sont pas nécessairement des artefacts de jeu (Genvo, 2011). La "*ludification*", traduction en français du terme "*gamification*" est un autre terme diffusé dans la littérature anglophone et dans les pratiques professionnelles de conception d'expériences ludiques (Zichermann et Cunningham, 2011). Le concept de gamification est défini comme un processus qui emploie la mécanique de jeu et la pensée de jeu pour engager l'utilisateur et lui permettre de résoudre des problèmes (Zichermann et Cunningham, 2011). Il est considéré distinct de celui de ludicisation, il se restreint à encourager et engager un individu dans toutes sortes d'activités en employant des structures de récompense et des renforcements positifs, tant disque le concept de ludicisation désigne d'une certaine manière une situation d'apprentissage qui exploite des caractéristiques ludiques (Genvo, 2012).

Enfin, l'expression "*Game Based Learning*" (apprentissage par le jeu), souvent précédée par le mot "*Digital*" (numérique), pour "*Digital Game Based Learning*" (Prensky, 2007), est la plus utilisée pour réunir l'ensemble des formes que peuvent prendre les pratiques d'exploitation de jeux numériques en éducation et de mise en place de situations d'apprentissage ludiques.

Nous allons maintenant présenter quelques efforts de conception et d'évaluation de jeux d'apprentissage, au travers de modèles de conception et d'analyse existants dans

1. Le terme "*situation*" a également été utilisé par Vygotsky (1967) pour désigner les jeux qui *permettent de développer chez l'enfant ses capacités d'abstraction et son imagination*.

la littérature. L'objectif étant de faire ressortir les attributs communs à ces modèles et de comprendre le jeu et sa relation à l'apprentissage.

1.3 CONCEPTION DE JEUX POUR L'APPRENTISSAGE

Il existe dans la littérature plusieurs modèles et grilles théoriques ou empiriques pour la conception et/ou l'analyse de jeux dédiés à l'apprentissage, définissant des principes clés permettant la caractérisation de ces derniers.

On peut citer le framework de DE FREITAS et OLIVER (2006) qui a pour objectif de fournir une démarche d'évaluation itérative permettant aux enseignants/formateurs de choisir le jeu à intégrer dans leur formation, aux évaluateurs et aux chercheurs de développer des mesures pour analyser efficacement des jeux et des simulations existantes et aux pédagogues de considérer un ensemble de facteurs pédagogiques spécifiques plus adaptés aux utilisateurs. Ce framework peut s'utiliser sur des jeux existants ou des jeux en cours de conception (DE FREITAS et OLIVER, 2006). Il est fondé sur quatre dimensions interdépendantes : la première considère le *contexte* du jeu et de l'apprentissage incluant des outils et des ressources spécifiques. La deuxième concerne les attributs relatifs à *l'apprenant* ou à un *groupe d'apprenants*, incluant leurs profils de compétences et de préférences. La troisième se rapporte à la *représentation interne de l'univers du jeu* ou de la simulation, qui inclut les interactions Homme-Machine, le niveau de l'immersion,...etc. La quatrième dimension se concentre sur les *processus d'apprentissage* et incite en particulier à réfléchir aux méthodes, théories et modèles d'apprentissage à utiliser. Ce modèle considère également l'importance du débriefing pour renforcer l'apprentissage.

Un autre exemple est le framework conceptuel de VAN STAALDUINEN et FREITAS (2011). Il considère une première dimension liée à l'*apprentissage* qui sert à spécifier les objectifs du jeu et le contenu d'apprentissage. La seconde dimension *éducation* invite à réfléchir sur le cycle d'apprentissage, qui regroupe le comportement, le feedback et l'engagement de l'apprenant. Elle repose en partie, sur le cycle d'apprentissage défini par GARRIS, AHLERS et DRISKELL (2002), que nous détaillerons plus loin (cf. s. 1.3.1.1) et sur le framework de DE FREITAS et OLIVER (2006). Enfin, la troisième dimension *évaluation* concerne le débriefing et le feedback du système qui permettent la construction de l'apprentissage résultant du jeu joué par l'apprenant dans son contexte de formation.

On peut également citer le modèle GOMII (Game Object Model), qui est un modèle théorique pour la conception de jeux éducatifs (AMORY, 2007). Il est conseillé de l'utiliser comme une check-list afin de vérifier les aspects conceptuels d'un jeu éducatif (AMORY, 2007). Ce modèle considère que le jeu impacte l'apprentissage à travers le visuel, l'expérience, la créativité, la découverte, la résolution de problèmes, la manipulation d'objets et la compétition entre participants.

Dans ce qui suit, nous détaillerons, dans un premier temps, quelques modèles de conception et d'analyse de jeux et d'apprentissage, qui explicitent un grand nombre de caractéristiques propres au jeu et à l'apprentissage par le jeu. Dans un second temps, nous détaillerons quelques grilles d'analyse et de conception, qui décrivent les facettes d'analyse du jeu et de l'apprentissage pouvant être exploitées lors de la conception de jeux.

1.3.1 *Modèles de jeux pour l'apprentissage*

1.3.1.1 *Le modèle Entrée-Processus-Résultat*

Le modèle Entrée-Processus-Résultat (Input-Process-Outcome) est un modèle de jeux et d'apprentissage qui rassemble des critères de conception de jeux à finalité éducative (Garris, Ahlers et Driskell, 2002). Il comprend un cycle de jeu incluant les jugements et le comportement de l'utilisateur, ainsi qu'un feedback impactant l'engagement et la motivation de ce dernier lors du jeu *play*. Il décrit également les objectifs d'apprentissage possiblement atteignables. Il se fonde sur une structure de jeu, dans laquelle un contenu éducatif est combiné avec certaines caractéristiques du jeu, donnant lieu à un jeu *play* cyclique et récursif, impliquant l'engagement de l'utilisateur. Ce qui implique la construction des objectifs d'apprentissage (Figure 2).

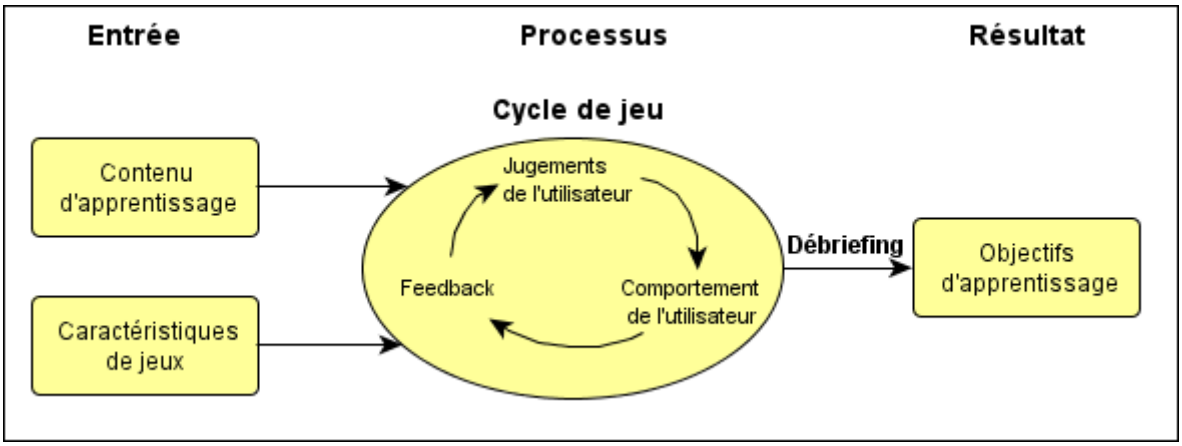


Figure 2 – Le modèle Entrée-Processus-Résultat (Garris, Ahlers et Driskell, 2002).

caractéristiques de jeu. Elles sont décrites dans ce modèle comme suit (Garris, Ahlers et Driskell, 2002) :

- *Fantaisie* : le jeu se caractérise par un monde imaginaire (thèmes, personnages, ...etc.), dans lequel les actions du joueur n'ont pas de conséquences sur le monde réel. La fantaisie aide à capter l'attention de l'apprenant et à maintenir sa concentration afin de l'immerger dans son activité.
- *Règles/Objectifs* : les règles du jeu décrivent sa structure et son objectif. Des objectifs spécifiques et précis permettent de maintenir l'attention et la motivation de l'apprenant, notamment à travers un feedback constant. Les règles doivent également être suffisamment permissives à l'égard des actions dans le jeu.
- *Stimuli sensoriels* : ce sont tous les stimuli visuels et auditifs qui sont perçus par les joueurs, tels que des effets sonores ou des animations. Ce sont des éléments qui renforcent la motivation et l'intérêt de l'apprenant.
- *Challenge* : le challenge est lié à un niveau de difficulté adaptée et croissant et à un objectif difficile à atteindre. Néanmoins, les objectifs doivent avoir un sens pour l'apprenant. Pour cela, les tâches inclues dans les scénarios doivent être liées aux compétences du joueur. La compétition et la coopération rendent également les objectifs plus claires.

— *Mystère* : ce facteur est lié à l'intérêt suscité par des sensations nouvelles que fournissent les stimuli et à la curiosité cognitive qui est un désir de connaissances. Ce facteur implique également l'introduction d'effets de surprise.
— *Contrôle* : la maîtrise du jeu aide à maintenir la motivation de l'apprenant et à favoriser son apprentissage. Le jeu engendre un sentiment de contrôle chez l'apprenant lorsque ce dernier est autorisé à choisir des stratégies et à prendre des décisions qui influencent directement le résultat final.

CYCLE DE JEU. Il s'agit du deuxième composant de ce modèle, qui est décrit comme le processus qui maintient la motivation de l'apprenant, dans lequel des cycles répétitifs sont mis en place, incluant les *jugements* de l'apprenant, son *comportement*, ainsi qu'un *feedback* du système en résultat aux actions de l'apprenant. Le cycle de jeu se concentre sur une série de dépendances : *afin d'arriver à un comportement souhaitable de l'apprenant, ce dernier doit avoir des réactions émotionnelles et cognitives souhaitables, résultant de son interaction avec le jeu qui lui répond via un feedback constant*. Les différents éléments qui constituent le cycle du jeu sont décrits comme suit :

— *Jugements de l'utilisateur* : ce sont les jugements subjectifs de l'utilisateur lors du jeu *play* quant à *l'intérêt* exprimé par l'apprenant pour l'activité entreprise dans le jeu et au *plaisir* ressenti lors du jeu *play*. Le sentiment de plaisir est lié au fait que les défis du jeu sont adaptés aux compétences de l'apprenant et à son implication dans les activités du jeu. L'implication de l'apprenant est en lien avec le niveau de son attention et de sa concentration et à son sentiment de confiance et de contrôle développé au fil du jeu *play*. Ce sentiment se développe car les conséquences des actions des apprenants et leurs échecs n'ont pas d'impacts sur le monde réel. Ce sentiment est également lié à la progressivité des niveaux de difficultés des différentes tâches constituant le jeu.
— *Comportement de l'utilisateur* : les jugements développés lors du jeu *play* déterminent le comportement de l'apprenant. Un apprenant motivé va s'engager et persévérer à la réalisation des activités. Il est intéressé et impliqué dans les activités et y consacre du temps.
— *Feedback* : les jugements individuels et le comportement de l'apprenant sont régulés par le feedback du système. Si le feedback indique que les objectifs sont constamment atteints, la motivation de l'apprenant diminue. Si le feedback indique, au contraire, que la performance de l'apprenant est en dessous de celle désirée, alors celui-ci a le choix entre deux options, abandonner l'activité ou doubler d'efforts afin d'atteindre la performance visée. De ce fait, le feedback évalue la progression de l'apprenant à travers les objectifs régissant le jeu, en impactent la motivation et l'implication du joueur.

DÉBRIEFING. Ce modèle souligne également l'importance du *débriefing* qui constitue le lien fondamental entre l'expérience du jeu et les résultats d'apprentissage. Le débriefing est décrit, ici, comme l'élément permettant de revoir et d'analyser les événements qui se sont produits dans le jeu, pour les transformer en expériences d'apprentissage.

OBJECTIFS D'APPRENTISSAGE. Les connaissances à acquérir par le jeu sont décrites dans ce modèle sur la base d'une classification hiérarchique des objectifs d'apprentissage :

— *Objectifs fondés sur les compétences* : incluent le développement de compétences techniques ou motrices.
— *Objectifs cognitifs* : incluent des *connaissances déclaratives*, en référence aux connaissances des faits et données requises pour l'élaboration de tâches. *Connaissances procédurales*, en référence à la connaissance de comment élaborer une tâche. *Connaissances stratégiques*, qui requièrent l'application de règles et de stratégies dans des situations générales ou nouvelles.
— *Objectifs affectifs* : réactions affectives incluant le sentiment de confiance, auto-efficacité, attitudes, préférences et dispositions de l'apprenant.

Nous allons maintenant décrire le modèle de jeu expérientiel de (KIILI, 2005) qui possède certaines similarités avec le modèle Input-Process-Outcome de (GARRIS, AHLERS et DRISKELL, 2002). Il considère au même titre que ce dernier, que l'apprentissage découle d'un processus cyclique résultant d'une expérience produite par les interactions avec le jeu.

1.3.1.2 *Le modèle de jeu expérientiel*

Le modèle de jeu expérientiel est un modèle de conception et d'analyse de jeux éducatifs (KIILI, 2005). Il est fondé sur l'expérience de *flow*, qui est décrite comme un état d'immersion complet d'un individu dans son activité (KIILI, 2005). L'objectif principal de ce modèle est de faciliter l'expérience du *flow* en liant le jeu *play* à un apprentissage expérientiel (Figure 3).

Ce modèle décrit l'apprentissage comme un processus cyclique à travers une expérience directe dans le jeu. Il définit l'apprentissage par le jeu comme une construction de structures cognitives suite aux interactions avec le jeu. Ce modèle souligne l'importance d'offrir au joueur un *feedback* permanent, des objectifs de jeu précis et des défis en adéquation avec le niveau de compétences du joueur (KIILI, 2005).

Les *challenges* se basent sur les objectifs pédagogiques et constituent le cœur du modèle. La fonction de ce dernier est de maintenir la motivation et l'engagement de l'élève en posant des défis adaptés qu'il relèvera en générant les solutions adéquates. La génération de solutions est subdivisée en génération d'idées dites *preinvatives* et en génération d'*idées*. Une idée *preinvative* fait référence à la créativité primaire. Elle est décrite comme une phase de jeu *play* non structurée, c'est-à-dire qu'elle est générée sans la prise en compte des contraintes du système, résultant sur des solutions nouvelles (KIILI, 2005). Suite à cette phase, le joueur développe des solutions en prenant en compte certaines contraintes et ressources disponibles dans le jeu. Après cela, le joueur teste ses solutions dans un cycle d'expérience. Le jeu doit être utilisable, avoir des objectifs précis et doit fournir un *feedback* approprié afin de faciliter l'expérience du *flow*. L'observation du feedback va aider le joueur à élaborer des solutions mieux adaptées au problème.

Ce modèle souligne également l'importance d'autres éléments liés au *flow* telle que l'*attention*. L'observation du feedback peut amener le joueur à la construction de schèmes et la découverte de nouvelles solutions au problème. Le fait de tester diffé-

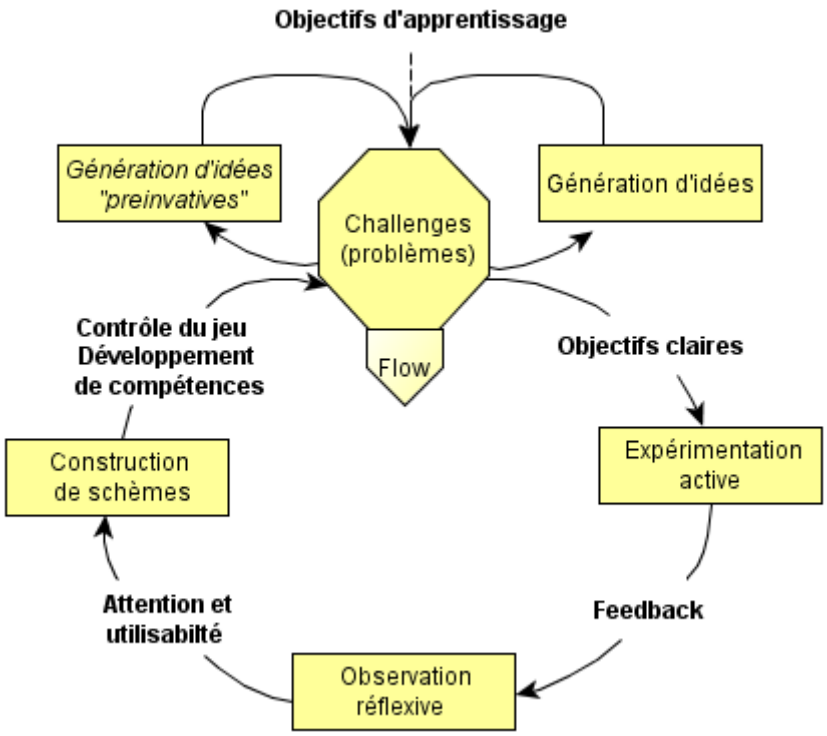


FIGURE 3 – Le modèle de jeu expérientiel (KIILI, 2005).

rentes solutions, peut amener le joueur à une situation de *contrôle* sur le jeu (KIILI, 2005).

Nous allons maintenant passer au modèle 3E de (SANCHEZ, 2013) qui définit un certain nombre d'attributs caractérisant le jeu selon un point de vue différent, c'est-à-dire en ce concentrant sur la situation ludique qui découle des interactions avec la structure du jeu.

1.3.1.3 *Le modèle 3E*

Le modèle 3E (*Enroll, Entertain, Educate*) est un modèle de conception de situations ludiques de jeux épistémiques numériques (SANCHEZ, 2013). Il regroupe un ensemble de critères de conception en considérant les éléments essentiels à l'implication de l'étudiant dans le jeu (*Enroll* ou "Motiver"), les éléments permettant le divertissement de l'apprenant (*Entertain* ou "Divertir") et les éléments essentiels à la construction des connaissances chez l'apprenant (*Educate* ou "Éduquer") (Figure 4).

ENROLL. Cette première dimension du modèle s'intéresse aux critères en lien avec la motivation et l'implication de l'apprenant dans le jeu qui sont décrits comme suit :

— *Sentiment de compétence* : augmente quand l'étudiant atteint des *objectifs précis*. Pour cela le jeu doit fournir des feedbacks fréquents permettant à l'apprenant de reconnaître ses réalisations. Cependant, le niveau de difficulté doit s'adapter au niveau de l'apprenant, mais peut croître au fur et à mesure que ce dernier avance dans son apprentissage.

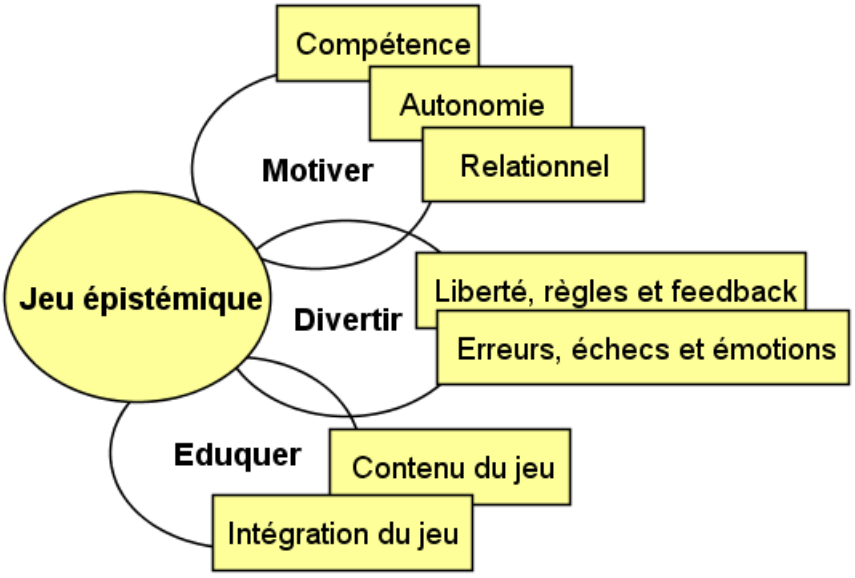


FIGURE 4 – Le modèle 3E (SANCHEZ, 2013).

— *Autonomie* : implique la liberté de réaliser des choix, de *prendre des décisions et de choisir une stratégie*. Cette attitude apparait quand l'apprenant accepte les défis du jeu et s'engage dans la résolution de problèmes, tout en restant libre de prendre des décisions. Cependant, cette liberté est limitée par les règles du jeu.
— *Le besoin de relationnel* : entraîne un besoin de compétition. Le fait de gagner des points et de pouvoir accéder à un niveau supérieur aide à maintenir la motivation. Le succès résulte également de la capacité des participants à *collaborer*.

ENTERTAIN. Les critères de la seconde dimension sont décrits comme suit :
— *liberté, règles et feedback* : la liberté sous-entend que l'apprenant est autorisé à prendre des décisions et à choisir une stratégie. Si le feedback est régulier, l'apprenant peut anticiper ses actions et s'adapter à la situation en appliquant de nouvelles stratégies. De ce fait, l'apprenant adapte ses actions et apprend à se comporter face aux contraintes du jeu.
— *Erreurs, échecs et émotions* : l'apprenant peut commettre des erreurs sans être sanctionné, il est au contraire encouragé à continuer. Le jeu doit également considérer les aspects émotionnels en lien avec l'humour, la qualité graphique de l'environnement du jeu, les interactions sociales, l'imagination et la curiosité. Les récompenses sont importantes dans le maintien de la motivation de l'apprenant.

EDUCATE. Cette dernière dimension repose sur les critères suivants :
— *Le contenu* : le jeu est un modèle d'une situation réelle matérialisée par un ensemble de concepts sélectionnés par le concepteur. Il est considéré comme étant une méta-activité ou une métaphore de la réalité. Cependant, le jeu doit représenter le domaine d'intérêt et inclure les connaissances appropriées.
— *Intégration du jeu dans le processus d'apprentissage (débriefing)* : la connaissance développée dans le jeu est implicite, le rôle de l'enseignant est donc crucial pour aider

les apprenants à prendre conscience de cette connaissance qu'ils transféreront à d'autres situations.

Dans ce qui suit nous présentons quelques grilles d'analyse réunissant un certain nombre d'attributs d'analyse, pouvant être considérés lors de la conception de jeux dédiés à l'apprentissage.

1.3.2 *Grilles d'analyse et de conception*

1.3.2.1 *La grille de notation d'*Anneta, Lamb *et* Stone *(2011)*

La grille de notation d'Anneta, Lamb et Stone (2011) est une grille de notation théorique pour l'analyse de jeux sérieux éducatifs. Elle repose sur un protocole construit sur la base de la théorie de l'apprentissage constructiviste et considère les aspects suivant : *Apprentissage par l'expérience* où le joueur s'engage activement dans le scénario du jeu en prenant des décisions. *Apprentissage inquisitif* où l'exploration et la découverte permettent d'atteindre le but global du jeu. *Sentiment d'auto-efficacité* que les jeux vidéos font croître en récompensant constamment le joueur et en l'encourageant à accéder à un niveau supérieur. *Objectifs bien définis*, composante essentielle qui distingue jeu d'une simulation. *Feedback constant* grâce auquel les jeux font croître la participation du joueur. *Apprentissage coopératif* qui améliore considérablement l'apprentissage comparativement à un apprentissage individuel.

La grille possède les composantes qui sont décrites comme suit (Anneta, Lamb et Stone, 2011) :

— *Prologue* : permet d'introduire de façon claire les objectifs du jeu, afin d'être facilement perçus par le joueur et d'assurer l'expérience du *flow*.
— *Tutoriel* : décrit l'interface utilisateur.
— *Interactivité* : quand le joueur interagit avec l'environnement du jeu, des événements sont déclenchés lui offrant un retour permettant de l'immerger et de l'amener à une situation de *flow*.
— *Feedback* : retour constant du jeu permettant au joueur de développer des stratégies et d'apprendre par l'erreur.
— *Identité* : la représentation de l'identité du joueur par un avatar peut faire croître son sentiment d'immersion.
— *Immersion* : la situation du *flow* qui se produit quand le joueur se retrouve totalement immergé dans son activité et perd momentanément la notion de temps et d'espace.
— *Manipulation* : lors des interactions, le joueur va manipuler des objets de l'environnement en réalisant un certain nombre d'actions. Ces manipulations doivent avoir un impact sur les résultats du jeu.
— *Difficulté croissante* : l'apprentissage doit être progressif. Pour cela les activités doivent être de difficulté croissante. Cela aide à maintenir la motivation de l'apprenant.
— *Règles* : le jeu sérieux a besoin de règles explicites ayant des conséquences bien définies.

— *Apprentissage fondé sur un feedback* : un feedback des activités effectuées par le joueur est important pour le processus d'apprentissage.
— *Objectifs d'apprentissage* : le jeu sérieux doit avoir des objectifs d'apprentissage bien précis dès la phase de conception afin d'être évalués.
— *Efficacité pédagogique* : le jeu est pédagogiquement efficace si l'étudiant arrive à comprendre les concepts considérés dans le jeu.
— *Textes informatifs* : le texte dans le jeu doit être concis et précis pour que le joueur puisse le retenir.
— *Feedbacks sonore et visuel* : si un feedback sonore est prévu, alors il ne doit pas surcharger le feedback visuel.

Nous allons maintenant présenter la grille d'indicateurs de qualité de Marfisi et al. (2012), qui fournit des directives conceptuelles suivant plusieurs facettes.

1.3.2.2 *La grille d'indicateurs de qualité de* Marfisi *et al. (2012)*

La grille proposée par Marfisi et al. (2012) réunit un ensemble d'indicateurs en lien avec la qualité pédagogique d'un jeu d'apprentissage, son potentiel ludique et son utilité dans le contexte de formation. Les indicateurs sont fournis comme une liste de conseils permettant d'analyser la qualité d'un jeu d'apprentissage lors de sa conception.

La grille comporte six facettes et sont décrites comme suit (Marfisi et al., 2012) :

objectifs pédagogiques. Regroupent les indices de qualité décrits comme suit :
— *Intégration des compétences cibles dans le scénario* : Cela permet de répondre aux objectifs pédagogiques prévus par le concepteur.
— *Activités pédagogiques adaptées à la formation* : la construction de compétences doit se faire à travers la réalisation de différentes activités dans lesquelles l'apprenant a un rôle actif comme enquêter, simuler, diagnostiquer, manipuler des outils, créer des objets...etc. Chaque activité doit être choisie de façon à ce qu'elle soit suffisamment adaptée à la formation.
— *Structuration des activités pédagogiques* : l'ordre dans lequel les activités se suivent doit être soigneusement planifié.

interactions. Il s'agit des interactions de l'apprenant avec d'autres acteurs humains (apprenants, ou formateurs) ou dispositifs technologiques (ordinateurs, tablettes,...etc). Cette facette regroupe les indicateurs suivants :
— *Qualité et pertinence des interactions* : le choix des interactions Homme-Machine et des interactions Homme-Homme doit favoriser l'apprentissage, augmenter le plaisir de l'apprenant et assurer l'acceptation du jeu par les formateurs.
— *Diversité et attractivité des interactions* : cela permet de promouvoir une pédagogie active reposant sur l'implication de l'apprenant dans son apprentissage.

problèmes et progression. Comprend les indicateurs décrits comme suit :
— *Motivation intrinsèque* : favorise l'apprentissage et augmente l'intérêt et le plaisir de l'apprenant dans le jeu.
— *Phases d'apprentissages régulières et constantes* : le jeu ne doit pas avoir de grandes phases dénuées d'activités pédagogiques.

— *Liberté d'action et stratégies de résolution* : les actions du joueur ne doivent pas avoir d'impact sur l'environnement réel (frivolité). Il faut également inclure des moments où les apprenants peuvent jouer librement en apprenant par essai-erreurs afin de favoriser leur autonomie et leur implication dans le jeu.
— *Challenge constant* : permet de capter l'attention de l'apprenant. Pour cela, le niveau de difficulté doit être bien adapté.
— *Reconnaissance de la progression du joueur dans le jeu* : peut être sous forme de scores, trophée, message de réussite, ou de déblocage d'un élément dans le scénario. Les émotions déclenchées quand les apprenants gagnent ou perdent ont des effets positifs sur leurs niveaux d'attention, leurs mémoires et leurs prises de décision.

DÉCORUM. Représente tous les éléments multimédias du scénario qui procurent du plaisir au joueur. Cette facette regroupe des indicateurs de qualité liés au choix du décorum en fonction du profil des apprenants et du contexte d'utilisation du jeu :
— *Pertinence du décorum* : il est préférable de choisir un lieu, une époque, des personnages et des missions dans lesquelles les compétences cibles ont un sens. Cela favorise le transfert des compétences à des situations réelles et l'acceptation du jeu par les formateurs.
— *Attractivité et originalité du jeu* : l'attractivité du jeu peut être améliorée par les effets visuels et sonores, l'humour ou l'originalité de l'histoire ou des graphismes. Il est aussi conseillé d'introduire des éléments de surprise dans le jeu pour garder l'apprenant en état actif et stimuler ses émotions.

CONDITIONS D'UTILISATION. Cette facette regroupe les indicateurs suivants :
— *Respect des contraintes liées aux aspects techniques et organisationnels du contexte de formation* : doivent être compatibles avec les activités de formation. Le nombre d'apprenants et de formateurs disponibles doit également être pris en compte.
— *Intégration du jeu au contexte de formation avec des débriefings* : pour que les apprenants arrivent à acquérir les compétences visées par le jeu et les intégrer dans leur formation globale, il est important d'inclure des phases de débriefing pendant lesquelles l'enseignant pourra revenir sur les activités du jeu, identifier les compétences qui ont été construites et présenter des situations réelles dans lesquelles ces compétences peuvent être réutilisées.

COÛT PRÉVISIONNEL. Concernant sa conception, sa réalisation, sa mise en œuvre et son utilisation. Cette facette regroupe les indices suivants :
— *Réutilisation des composants logiciels* : cela peut réduire le coût de développement.
— *Clarté et détails des spécifications* : le temps de réalisation dépend du niveau de précision et de détails des spécifications.
— *Erreurs de connexion dans le scénario* : leur présence pourrait ralentir le processus de création du jeu.

Afin d'affiner notre compréhension du jeu et de situer l'apprentissage, nous allons décrire les pratiques d'évaluation de jeux existantes. Le but étant d'identifier les facettes du jeu qui sont évaluées et les démarches d'évaluation courantes.

1.4 ÉVALUATIONS DES JEUX

1.4.1 *Utilisabilité, utilité et acceptabilité*

Afin de simplifier les relations complexes entre les fonctionnalités d'un système informatique et leur intérêt pour l'utilisateur, SENACH (1990) définit le processus d'évaluation sur la base de deux dimensions principales : celle qui concerne *l'utilité* du logiciel, elle souligne son adéquation aux objectifs de haut niveau de l'utilisateur et celle en lien avec *l'utilisabilité* du logiciel, qui est sa qualité et sa capacité à permettre à l'utilisateur d'atteindre facilement ses objectifs.

Dans le contexte des Environnements Informatiques pour l'Apprentissage Humain (EIAH), les objectifs d'utilisation de haut niveau concernent des objectifs d'apprentissage. De ce fait, l'évaluation de l'utilité concerne l'impact de l'enseignement dispensé par cet EIAH sur les connaissances et les compétences des apprenants (BARBEL, 2003 ; NOGRY, JEAN-DAUBIAS et OLLAGNIER-BELDAME, 2004), quant à l'évaluation de l'utilisabilité, elle concerne les aspects ergonomiques (BARBEL, 2003) et vérifie l'adéquation entre la manière dont une tâche est réalisée par un apprenant et les capacités cognitives de cet apprenant (NOGRY, JEAN-DAUBIAS et OLLAGNIER-BELDAME, 2004).

En plus de l'utilisabilité et de l'utilité d'un EIAH, TRICOT et al. (2003) considèrent une troisième dimension qui est son *acceptabilité* qu'ils définissent comme la valeur de la représentation mentale comprenant les attitudes et les opinions plus au moins positives de cet EIAH, de son utilité et de son utilisabilité. La valeur de cette représentation conditionne l'utilisation de l'EIAH.

Dans le contexte d'apprentissage par le jeu, l'acceptabilité recoupe en partie le contexte d'utilisation du jeu, elle dépend du jugement de l'étudiant, de l'enseignant et de l'institution porté sur sa valeur (SANCHEZ, 2011). L'utilisabilité reflète la possibilité d'utiliser le jeu dans un contexte d'apprentissage, elle dépend des aspects ergonomiques et de la qualité de son interface utilisateur (WARREN, JONES et LIN, 2011). Les tests d'utilisabilité permettent de s'assurer de la jouabilité du jeu et de l'adéquation de son contenu avant son déploiement en classe. Ils permettent d'examiner les éléments qui influencent la capacité de l'apprenant à réussir les activités du jeu. L'utilité est centrée sur les aspects didactiques du jeu, son évaluation permet de vérifier l'impact du jeu sur le processus d'apprentissage et ses retombées pédagogiques (SZILAS et SUTTER WIDMER, 2013).

Nous donnons dans ce qui suit, les pratiques d'évaluation répandues dans la littérature, qui se distinguent selon que l'évaluation intervienne au cours du processus de conception ou à la fin de ce processus.

1.4.2 *Méthodes d'évaluation analytiques et empiriques*

Afin de mesurer l'utilisabilité d'une Interface Homme-Machine (IHM), SENACH (1990) souligne deux approches d'évaluation : *une approche analytique* qui consiste à étudier l'interface selon un ensemble de référents définis par des experts pour contrôler qu'elle possède bien certaines qualités et *une approche empirique* qui consiste à recueillir les données relatives au comportement de l'utilisateur afin de déterminer les difficultés

d'utilisation et de développer des solutions les réduisant. Cette distinction a ensuite été reprise par Nogry, Jean-Daubias et Ollagnier-Beldame (2004) dans l'évaluation de l'utilisabilité des EIAH, en insistant sur l'adaptation des différentes méthodes aux spécificités des EIAH. De la même façon, Tricot et al. (2003) admettent cette distinction pour l'évaluation de l'utilisabilité, l'utilité et l'acceptabilité des EIAH, en précisant qu'il s'agit de deux approches strictement distinctes et complémentaires.

L'évaluation d'un jeu d'apprentissage peut également être réalisée par l'emploi de ces deux méthodes d'évaluation, en s'étalant sur ses phases de conception et de développement (Djelil, 2014 ; Djelil, Sanchez et al., 2014).

1.4.2.1 *Évaluation analytique*

Dans le contexte des IHMs, l'évaluation analytique consiste à diagnostiquer la qualité ergonomique de l'interface sans enregistrer les données relatives à son utilisation (Senach, 1990). Elle est définie comme une *évaluation a priori* des caractéristiques globales de l'interface. Elle est alors réalisée en comparant cette dernière à un modèle de référence décrivant les propriétés d'une bonne interface. Senach (1990) précise que la difficulté d'une telle analyse réside dans l'identification des dimensions pertinentes, la construction des échelles de mesure et l'intégration dans une appréciation globale de résultats caractérisant l'interface selon des points de vue très différents.

Nous avons par ailleurs soutenu l'idée que dans le cadre de jeux d'apprentissage, l'utilisation des grilles et modèles conceptuels (cf. s. 1.3) comme référentiels pour l'évaluation des différents aspects conceptuels de ces derniers, constitue une évaluation analytique du jeu (Djelil, Sanchez et al., 2014). Bien que l'expression "*évaluation analytique*" n'ait pas été utilisée, à notre connaissance, dans le contexte de jeux d'apprentissage, les pratiques sont bien présentes dans la littérature. Ainsi, Di Loreto et Gouaich (2010) explique que les résultats de l'application des framework conceptuels de jeux, peuvent être utiles aux concepteurs afin de déterminer les limites fonctionnelles d'un jeu avant son implémentation. Ces modèles et grilles conceptuelles peuvent être fournis aux concepteurs sous forme d'une liste de conseils de conception et être ensuite exploités dans l'évaluation de la qualité des différents aspects du produit final (De Freitas et Oliver, 2006 ; Marfisi et al., 2012 ; Marne, Huynh-Kim-Bang et Labat, 2011).

1.4.2.2 *Évaluation empirique*

L'évaluation empirique consiste à recueillir des données relatives à l'utilisation du système. Elle nécessite l'existence de maquettes, de prototypes ou de système final et la présence d'utilisateurs (Nogry, Jean-Daubias et Ollagnier-Beldame, 2004). D'après Senach (1990), l'évaluation empirique vise soit à diagnostiquer les usages du système, soit à effectuer des tests de conception afin de remédier aux défauts de conception éventuels.

Dans le cadre de jeux d'apprentissage, les données recueillies lors de diagnostics d'usage traduisent de façon structurée les pratiques permettant d'identifier les modalités d'exploitation réelle du jeu. Les techniques de recueil utilisées aujourd'hui, sont les traces d'interaction numériques, les interviews directes, les questionnaires d'utilisation, les notes d'observations directes et les enregistrements audios et vidéos (Dunleavy et

Simmons, 2011). Les interviews et les questionnaires d'utilisation permettent de complémenter et de valider les résultats expérimentaux par le recueil d'appréciations subjectives auprès des utilisateurs. Ce sont le plus souvent des enquêtes de satisfaction et d'utilité qui sont conduites de façon prospective afin de contrôler la qualité du jeu évalué (Navarro et Van Der Hoek, 2007).

Cependant, la présence d'utilisateurs n'est pas la seule façon de recueillir des données expérimentales relatives à l'usage du jeu. La simulation d'environnements de tests est également possible. Dans ce contexte, un simulateur de comportements d'utilisateurs nommé SIMCA "SIMulateur de Comportements d'Apprenants" a été conçu pour vérifier la qualité pédagogique d'un jeu d'entreprise dès sa phase de conception (George, Titon et Prévot, 2005). Les échanges entre le jeu et le simulateur sont sauvegardés en logs afin d'être analysés en vue d'aider les concepteurs à identifier des corrections nécessaires avant l'utilisation réelle du jeu.

Les données expérimentales recueillies peuvent servir au diagnostic des compétences des apprenants (Oulhaci et al., 2013), à l'évaluation de l'efficacité du jeu d'apprentissage (Loh, 2012) et à la détection de ses disfonctionnements éventuels (R. Brown, 2011 ; Manin, George et Prévot, 2006). Il s'agit d'une évaluation empirique du jeu d'apprentissage.

L'évaluation est définie également selon un autre point de vue, selon qu'elle intervient lors de la construction des apprentissages ou à l'issue de l'activité d'apprentissage.

1.4.3 *Évaluation formative et sommative de l'apprentissage*

L'analyse et l'interprétation de données empiriques pour l'évaluation de l'apprentissage peut se faire soit au cours de l'activité d'apprentissage, soit à la fin de l'activité répondant ainsi à des objectifs d'évaluation différents. On parle alors *d'évaluation formative ou d'évaluation sommative* de l'apprentissage (Guyot, 1978), marquant ainsi des objectifs d'évaluation distincts.

1.4.3.1 *Évaluation formative*

L'évaluation formative tend d'apporter des informations sur les acquis en cours de construction et de situer la progression de l'apprenant par rapport à un objectif donné (Guyot, 1978). Elle permet également d'identifier les problèmes rencontrés par l'apprenant dans son apprentissage (Guyot, 1978).

On peut citer à titre d'exemple le système PeTRA (Performance TRacing Assistant), qui est un outil d'évaluation formative de jeux d'apprentissage (Loh, 2012). Cet outil offre une interface de visualisation interactive à la fois pour les formateurs, les apprenants et l'administrateur, leur permettant de visualiser des données pertinentes relatives aux différentes actions des apprenant en cours d'utilisation d'un jeu d'apprentissage. Ce système collecte, analyse et visualise en temps réel les traces d'interaction d'un jeu. Il est essentiellement utilisé par les formateurs afin d'analyser les actions du joueur en plein apprentissage. Le temps d'accomplissement des différentes tâches pour atteindre les objectifs d'apprentissage est rapporté et comparé à celui des experts. Ce système

permet également à l'apprenant d'auto-évaluer ses performances. D'un autre côté, l'administrateur peut s'intéresser à évaluer la performance et l'efficacité du jeu utilisé.

1.4.3.2 *Évaluation sommative*

L'évaluation sommative a pour but de tester la compréhension, la mémorisation et la maîtrise des apprentissages après leur enseignement (Guyot, 1978).

Dans le cadre du jeu d'apprentissage, les pratiques d'évaluation sommative sont nombreuses. Un exemple d'évaluation sommative dans un jeu d'apprentissage, est le jeu de labyrinthe collaboratif "Chase the Cheese" (Collazos et al., 2002). Ce jeu intègre un outil de collecte et d'analyse de traces d'interaction afin d'évaluer le processus d'apprentissage collaboratif en calculant des indicateurs de performance de groupe.

Dans le cadre du projet SIMFOR (Oulhaci et al., 2013) une évaluation sommative de l'apprentissage individuel et collectif est réalisée grâce à un système multi-agents qui collecte et traite les données relatives aux apprenants afin d'établir un diagnostic des compétences acquises et des compétences qui restent à acquérir.

Nous allons maintenant passer aux méthodes et techniques d'évaluation de l'apprentissage les plus répandues dans la littérature.

1.4.4 *Analyse de l'apprentissage*

L'analyse de l'apprentissage, expression traduite de l'anglais "*Learning analytics*", est définie comme la collecte, l'analyse et la production de rapports de données sur les apprenants et leurs contextes d'apprentissage, dans l'objectif de comprendre et d'optimiser l'apprentissage et l'environnement dans lequel il se produit (Siemens et Long, 2011). Il est en lien avec l'analyse de traces d'interaction, l'analyse de conversations, la recherche de modèles prédictifs et les algorithmes de fouilles de données et de classifications. Il s'intéresse également à l'adaptation de contenus pédagogiques sur la base de comportements d'apprenants, incluant l'usage de ressources numériques et les interactions entre apprenants (Siemens et Long, 2011).

Les méthodes de recherche largement adoptées dans cette discipline sont essentiellement de trois types : méthodes *qualitatives* incluant l'analyse de contenu, méthodes *quantitatives* incluant la vérification d'hypothèses par des tests statistiques sur des données collectées à travers des traces d'interaction et de questionnaires et méthodes *mixtes* combinant les deux types de méthodes qualitatives et quantitatives (Dawson et al., 2014). Les méthodes quantitatives semblent être dominantes, elles sont utilisées à des fins d'évaluation et de validation de questions de recherche et se caractérisent par la production de rapports de données issues de statistiques descriptives (Dawson et al., 2014).

Dans le contexte de jeux d'apprentissage, les pratiques d'évaluation sont imprégnées des méthodes se situant dans le domaine d'analytique de l'apprentissage. Les méthodes d'analyse les plus utilisées sont statistiques. Il existe également des méthodes d'analyse automatiques, qui reposent sur des algorithmes statistiques et d'intelligence artificielle.

1.4.4.1 *Analyse statistique*

L'analyse statistique est très employée dans l'évaluation de jeux pour l'apprentissage. Les pratiques existantes tentent d'établir des diagnostics d'usage, de mesurer l'impact de l'introduction du jeu dans la situation d'apprentissage (Pfahl et al., 2003 ; Von Wangenheim, Thiry et Kochanski, 2009), et de comparer son potentiel avec celui des méthodes d'enseignement traditionnelles (Rosas et al., 2003).

Le recours aux algorithmes statistiques pour l'évaluation de l'apprentissage dans les jeux, implique souvent des études comparatives effectuées sur deux groupes d'apprenants, un groupe *expérimental* ayant manipulé le jeu à évaluer et un groupe *contrôle* qui n'est pas concerné par l'utilisation du jeu. Les études comparatives donnent alors des indications sur l'utilisabilité, l'utilité et l'acceptabilité du jeu, à travers la vérification de son efficacité sur l'apprentissage, l'impact des interactions entre apprenants sur l'apprentissage, la motivation des apprenants et leur intérêt pour le jeu évalué (González-González et al., 2013 ; Papastergiou, 2009). Un exemple d'analyse statistique à travers une étude comparative a été élaboré dans l'évaluation du jeu éducatif *X-MED*, dont l'objectif pédagogique est de renforcer les concepts de mesure des systèmes logiciels (Von Wangenheim, Thiry et Kochanski, 2009). Pour réaliser cette évaluation deux hypothèses de tests statistiques ont été formulées afin de répondre à deux questions de recherche principales. La première concerne la différence de performances entre deux groupes d'étudiants, un groupe expérimental et un groupe contrôle. La seconde question est liée aux perceptions subjectives des étudiants quant à la pertinence du contenu, l'adéquation, le degré de difficulté, le déroulement, la méthode et la durée de l'apprentissage. Les données d'analyse ont été recueillies en effectuant des pré-tests et post-tests au travers de questionnaires et en exploitant des traces d'interaction. Un test statistique a été alors effectué afin de comparer la différence en termes de performance entre les deux groupes.

Cette démarche expérimentale est très répandue dans la littérature traitant de l'évaluation de jeux d'apprentissage. Il s'agit de trouver des réponses avec le moins d'erreurs possible à certaines questions de recherche relatives à l'utilité et à l'utilisabilité du jeu par le biais de tests statistiques. C'est le cas de l'étude comparative réalisée sur le jeu *Metalloman*, dédié à l'apprentissage de la physiologie destiné aux étudiants de premier cycle universitaire (Wong et al., 2007). Cette étude avait pour objectif de déterminer l'impact des interactions et la richesse des médias déployés dans le jeu sur l'apprentissage. Un autre exemple concerne l'étude de l'apport d'un jeu vidéo à l'apprentissage de la génétique et sa contribution au maintien de l'engagement d'élèves. (Annetta et al., 2009).

Par ailleurs, l'analyse statistique n'implique pas toujours des études comparatives. On peut citer les études réalisées sur le jeu Tamagocours, un jeu destiné à l'apprentissage des règles juridiques qui encadrent l'usage des ressources numériques dans un contexte éducatif (Sanchez et Emin-Martinez, 2014 ; Sanchez, Emin-Martinez et Mandran, 2015). Dans ces études, l'analyse des usages du jeu Tamagocours s'est traduit par l'analyse de traces d'interaction numériques et de conversations entre apprenants. Ces études ont conduit à l'identification de stratégies d'apprenants par la recherche de patterns d'actions, ainsi que de strates de jeu dans lesquelles les appre-

nants construisent leur apprentissage, en employant une méthode d'analyse statistique qui combine une analyse factorielle et une méthode de classification de données.

1.4.4.2 *Analyse automatique*

L'analyse automatique est très peu répandue dans l'évaluation de jeux d'apprentissage (Thomas, Labat et al., 2012). En effet, ce type d'analyse implique l'intégration de modules supplémentaires au sein de la structure interne du jeu et nécessite, par conséquent, des moyens de développement supplémentaires (Thomas, Labat et al., 2012).

Cependant, il existe quelques modèles d'analyse automatique de l'apprentissage qui ont été appliquées pour l'évaluation de l'apprentissage par le jeu. Le modèle Evidence Centered Design (ECD) (Mislevy et Riconscente, 2005), conçu pour la spécification formelle de l'évaluation de l'apprentissage, est l'un des modèles les plus utilisés (Rupp et al., 2010 ; Shaffer et al., 2009 ; Shute, L. Rieber et Van Eck, 2011).

Le modèle ECD permet la spécification du processus d'évaluation (Mislevy et Riconscente, 2005). Il repose essentiellement sur un framework conceptuel d'évaluation, appelé Conceptual Assessment Framework (CAF) qui spécifie les relations entre les déductions que l'évaluateur souhaiterait établir à propos de l'apprenant et d'éléments à observer, afin d'obtenir des inférences et de définir les situations en lien avec ces inférences (Mislevy et Riconscente, 2005). CAF est organisé sur la base de trois modèles principaux : un *modèle de l'apprenant* qui regroupe les connaissances et compétences de l'apprenant à évaluer, *un modèle d'évidence* contenant les composants d'évaluation concernant les productions de l'apprenant avec un modèle d'estimation des éléments d'apprentissage à évaluer et *un modèle de tâches* qui décrit l'environnement et la situation dans laquelle l'apprenant évolue (Mislevy et Riconscente, 2005). Ces trois modèles sont combinés par un *modèle d'assemblage* et le résultat d'évaluation est retourné par *un modèle de représentation*.

L'utilisation de l'ECD permet de vérifier la fiabilité des résultats obtenus par les apprenants et de mesurer les erreurs dues à certains paramètres non observées dans la situation d'apprentissage (Rupp et al., 2010). Un exemple d'implémentation du modèle CAF est le système ENA (Epistemic Network Analysis) (Shaffer et al., 2009), qui utilise ce modèle pour reconstituer un réseau des connaissances relatives aux interactions des apprenants avec le jeu. ENA fournit une analyse des compétences et de la progression des apprenants (Shaffer et al., 2009).

Il existe d'autres outils d'analyse automatique de l'apprentissage par le jeu. On peut citer les travaux de Thomas, Labat et al. (2012), dans l'élaboration d'un outil d'analyse qui fournit aux enseignants un support d'analyse des compétences et des progressions des apprenants. Cet outil repose sur un réseau de Petri qui décrit les actions de l'expert dans la résolution de problèmes et une ontologie du domaine et des actions du jeu définissant les liens entre les différentes actions du jeu et les compétences du domaine. Cet outil a été utilisé dans l'évaluation du jeu Ludiville dont l'objectif d'enseignement concerne le montage de dossiers de financement immobilier (Thomas, Yessad et Labat, 2011). De la même façon Koenig et al. (2009) ont développé un framework pour l'analyse de données d'usages de serious games, reposant sur une ontologie du domaine, les réseaux bayésiens et des outils d'analyse pour la représentation des actions d'appre-

nants. Cette représentation permet à la fois une évaluation formative et sommative des compétences des apprenants.

1.5 Synthèse : apprentissage par le jeu, faut-il considérer la situation ou plutôt l'artefact ?

1.5.1 *Vers une compréhension du jeu : jeu-game versus jeu-play*

Les modèles et grilles de conception et d'analyse de jeux présentés dans ce chapitre, donnent une liste d'attributs qui permettent de distinguer une partie du jeu qui est *l'artefact, la ressource ou la structure ludique* et une autre partie du jeu, qui est *la situation qui se met en place lorsque l'apprenant interagit avec cet artefact*. Cette distinction préalablement évoquée au début de ce chapitre (cf. s. 1.2), adoptée par quelques auteurs (Henriot, 1969 ; Sanchez, Emin-Martinez et Mandran, 2015 ; Shaffer, 2006) n'est pas explicitée par ces modèles, mais peut être confirmée en analysant les attributs qu'ils définissent et les relations existantes entre ces attributs.

La partie du jeu qui est l'artefact est souvent désignée dans ces modèles par l'expression *univers de jeu* (De Freitas et Oliver, 2006), *environnement de jeu* (Anneta, Lamb et Stone, 2011 ; Marfisi et al., 2012), ou *structure de jeu* (Garris, Ahlers et Driskell, 2002), renvoyant ainsi à la structure interne du jeu, dans laquelle un contenu éducatif est combiné avec des caractéristiques ludiques. Cette partie désignée par *jeu-game* (Sanchez, Emin-Martinez et Mandran, 2015) regroupe les composants interactifs du jeu grâce auxquels l'apprenant va construire son apprentissage lorsqu'il accepte d'interagir.

Quand l'apprenant accepte de jouer, l'interaction avec l'artefact induit la mise en place d'une situation, dans laquelle l'apprenant adopte une attitude ludique et développe des émotions et des sentiments comprenant notamment, un sentiment de plaisir lié au divertissement que procure l'expérience du jeu. L'ensemble des actions et réactions de l'apprenant constitue une seconde catégorie d'attributs définis dans ces modèles. Cette situation est appelée *jeu-play* dans l'intention de la distinguer du *jeu-game* (Sanchez, Emin-Martinez et Mandran, 2015). Souvent, elle est désignée simplement par "*play*" [2] (Garris, Ahlers et Driskell, 2002). Souvent, les actions formulées par l'apprenant ne se limitent pas qu'aux actions réalisée sur le jeu-game, mais incluent également des actions à destination d'autres participants [3]. Certains modèles insistent sur l'importance des interactions entre apprenants qui se mettent en place dans des tâches collaboratives ou compétitives (Amory, 2007 ; Sanchez, 2013).

Une troisième catégorie d'attributs identifiés dans ces modèles concerne le contexte d'exploitation du jeu et l'inclusion d'une phase de débriefing (De Freitas et Oliver, 2006 ; Marfisi et al., 2012 ; Sanchez, 2013 ; Van Staalduinen et Freitas, 2011). Cette phase dite, *phase d'institutionnalisation* signe la sortie du jeu-play : *elle se traduit par un changement de statut des connaissances qui, validées par l'enseignant, passent du statut d'ac-*

2. L'ambiguïté du terme *jeu* n'étant pas existante en langue anglaise, le *jeu-play* est désigné seulement par *play*.

3. Dans les travaux de Sanchez, Emin-Martinez et Mandran (2015) le *jeu-play* se subdivise en deux strates : *jeu-play1* qui décrit l'espace d'interactions entre l'apprenant et le *jeu-game* et *jeu-play2* qui comprend les interactions entre apprenants.

tions implicites dans le jeu à celui de savoirs explicites et institutionnalisés, susceptibles d'être transférées à d'autre situations d'apprentissage (Sanchez, Emin-Martinez et Mandran, 2015).

Dans ce qui suit, nous rappelons les attributs identifiés dans les différents modèles en les assignant aux différentes parties qui constituent une situation d'apprentissage par le jeu, dont le schéma général[4] est donné en Figure 5. Ce schéma est adapté du schéma de Sanchez, Emin-Martinez et Mandran (2015).

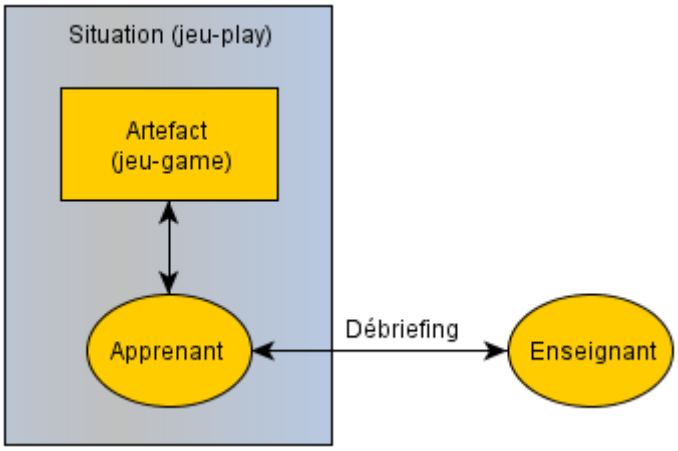


Figure 5 – Schéma général d'une situation d'apprentissage par le jeu adapté du schéma de Sanchez, Emin-Martinez et Mandran (2015).

1.5.1.1 *Attributs associés au jeu-game*

Les attributs associés au jeu-game décrits par la majorité des auteurs sont regroupés dans le Tableau 1. Ils sont décrits comme suit :

— *Objectifs et règles* : les objectifs du jeu sont transposés aux objectifs d'apprentissage ciblés par le concepteur (Marfisi et al., 2012). La réalisation de ces objectifs est encadrée par un ensemble de règles qui sont des contraintes à respecter et qui illustrent les concepts et notions constituant le domaine d'apprentissage (Garris, Ahlers et Driskell, 2002 ; Kiili, 2005).

— *Modalités de représentation* : c'est la représentation du contenu interactif de jeu. Cette représentation peut être graphique, textuelle, sonore et peut inclure des animations, des avatars ...etc. Elle peut employer des métaphores ou des narrations pour représenter des concepts abstraits (Dondlinger, 2007). Ces modes de représentation sont choisis de façon à stimuler l'apprenant afin d'agrandir son intérêt et sa motivation pour les activités du jeu (Garris, Ahlers et Driskell, 2002).

— *Difficulté progressive et adaptée* : les activités du jeu sont présentées à l'apprenant sous forme de défis (challenges) de complexité croissante (Garris, Ahlers et Driskell, 2002), afin d'assurer un apprentissage progressif tout en maintenant la motivation de l'apprenant (Anneta, Lamb et Stone, 2011).

4. Ce schéma met l'accent sur les interactions entre l'apprenant et le jeu-game et ne tient pas compte des interactions entre apprenants.

— *Échecs et erreurs* : les erreurs commises par l'apprenant sont sans conséquences réelles et ne sont pas sanctionnées, l'apprenant est autorisé à réessayer et à réfléchir sur l'origine de ses échecs (Marfisi et al., 2012 ; Sanchez, 2013).

	Objectifs et règles	Modalités de représentation	Difficulté progressive et adaptée	Échecs et erreurs
Descripteurs	règles bien définies et progression vers les objectifs	contenu interactif, graphiques, sons, animations, métaphores et narrations	défis de difficulté croissante	frivolité des échecs et erreurs
Recherches associées	Amory (2007), Anneta, Lamb et Stone (2011), De Freitas et Oliver (2006), Di Loreto et Gouaich (2010), Dondlinger (2007), Garris, Ahlers et Driskell (2002), Kiili (2005), Marfisi et al. (2012), Sanchez (2013) et Van Staalduinen et Freitas (2011)	Amory (2007), Anneta, Lamb et Stone (2011), De Freitas et Oliver (2006), Di Loreto et Gouaich (2010), Dondlinger (2007), Garris, Ahlers et Driskell (2002), Kiili (2005), Marfisi et al. (2012), Sanchez (2013) et Van Staalduinen et Freitas (2011)	Amory (2007), Anneta, Lamb et Stone (2011), Di Loreto et Gouaich (2010), Dondlinger (2007), Garris, Ahlers et Driskell (2002), Kiili (2005), Marfisi et al. (2012), Sanchez (2013) et Van Staalduinen et Freitas (2011)	Anneta, Lamb et Stone (2011), Garris, Ahlers et Driskell (2002), Marfisi et al. (2012), Sanchez (2013) et Van Staalduinen et Freitas (2011)

Tableau 1 – Attributs associés au jeu-game.

1.5.1.2 *Attributs associés au jeu-play*

Le jeu-play est souvent représenté par les auteurs par un cycle d'actions-rétroactions qui se met en place lorsque l'apprenant interagit avec le jeu-game (Garris, Ahlers et Driskell, 2002 ; Kiili, 2005). Certains auteurs parlent simplement d'*interactions*, qui font très souvent référence aux interactions de l'apprenant avec le jeu-game, mais peuvent désigner aussi les interactions entre apprenants (Dondlinger, 2007 ; Marfisi et al., 2012).

L'interaction entre l'apprenant et le jeu-game est préconisée au niveau des concepts d'apprentissage et doit susciter l'intérêt de l'apprenant pour le jeu. Elle est constituée des actions formulées par l'apprenant sur le jeu-game et des réponses du jeu-game aux actions de l'apprenant (feedback). Elle est toujours suivie de nombreux effets sur le comportement de l'apprenant, qui se répercutent de nouveau sur l'interaction elle même.

ACTIONS DE L'APPRENANT. Les actions formulées par l'apprenant peuvent être de plusieurs types :

— *Exploration* : c'est le fait de chercher des éléments dans le jeu-game afin de découvrir ses différents composants (Amory, 2007 ; Di Loreto et Gouaich, 2010).
— *Manipulation* : c'est le fait d'agir sur les composants du jeu-game afin d'accomplir un tâche donnée (Kiili, 2005 ; Van Staalduinen et Freitas, 2011).
— *Observation* : permet d'interpréter et de comprendre les feedbacks retournés par le jeu-game (Kiili, 2005 ; Marfisi et al., 2012).
— *Communication* : concerne la communication entre participants dans le cas où les activités sont coopératives ou compétitives (Amory, 2007 ; Dondlinger, 2007).

FEEDBACK. Il s'agit de la réponse du jeu-game aux actions de l'apprenant. C'est un retour qui aide l'apprenant à comprendre les règles du jeu, à prendre des décisions et à développer des stratégies pour résoudre les défis du jeu (ANNETA, LAMB et STONE, 2011), et donc de développer les connaissances et les compétences visées dans le jeu (GARRIS, AHLERS et DRISKELL, 2002). Cette dimension du jeu relève de l'évaluation formative qui sert à informer l'apprenant de ses acquis et de sa progression dans le jeu (cf.s. 1.4.3.1).

RÉACTIONS ET COMPORTEMENTS DE L'APPRENANT. Il s'agit de la troisième composante du jeu-play qui est constituée de l'ensemble des émotions, sentiments et attitudes qui sont générées chez l'apprenant lors de son interaction avec le jeu-game. Ces réactions sont directement liées aux caractéristiques du jeu-game. Elles impliquent le développement des connaissances et des compétences visées par la situation de jeu. Le Tableau 2 réunit l'ensemble des réactions et comportements identifiés dans les différents modèles. Ils sont décrits comme suit :

- *Auto-efficacité et contrôle* : ce sentiment se développe chez l'apprenant quand il réussit à lever les défis du jeu, dans la mesure où il est autorisé à prendre des décisions (DONDLINGER, 2007). Cela procure chez l'apprenant un sentiment de confiance et de contrôle qui aide à maintenir sa motivation (GARRIS, AHLERS et DRISKELL, 2002 ; KIILI, 2005).
- *Divertissement* : il s'agit du plaisir ressenti lors de son interaction avec le jeu-game. Il découle de l'appréciation des interactions et des modalités de représentation employées dans le jeu-game (GARRIS, AHLERS et DRISKELL, 2002).
- *Immersion et flow* : l'immersion est sous-jacente aux caractéristiques du jeu-game, notamment, à la qualité des modalités de représentation du jeu-game. Elle se produit quand l'apprenant est totalement submergé dans le jeu. Le *flow* est décrit comme le plus haut niveau d'immersion (ANNETA, LAMB et STONE, 2011).
- *Prise de décision et développement de stratégies* : lors de l'interaction de l'apprenant avec le jeu-game, il est amené à prendre des décisions et à développer des stratégies dans la résolution des défis du jeu (SANCHEZ, 2013). Il est libre d'agir et de tester ces choix dans le respect des règles du jeu.
- *Intérêt et motivation* : l'intérêt est le jugement positif porté par l'apprenant sur la qualité du jeu. Cela est lié aux caractéristiques du jeu-game et peut être maintenu par un sentiment de satisfaction lié aux connaissance et compétences apprise dans le jeu (GARRIS, AHLERS et DRISKELL, 2002). Cela induit l'engagement et l'implication de l'apprenant pour le domaine de connaissance.

1.5.1.3 *Contexte d'apprentissage et débriefing*

La majorité des auteurs insistent sur le *contexte d'apprentissage*, qui est perçu comme une partie intégrante du jeu (DE FREITAS et OLIVER, 2006 ; MARFISI et al., 2012 ; SANCHEZ, 2013). Le contexte doit être connu avant la mise en œuvre du jeu, afin d'assurer son intégration dans la formation. Il regroupe les conditions technologiques et organisationnelles d'exploitation du jeu, tels que le lieu de formation, le temps alloué à la formation, l'organisation du programme d'enseignement, le recours à des supports de formation complémentaires, la présence de formateurs, ...etc.

	Auto-efficacité et contrôle	Divertissement	Immersion et *flow*	Prise de décision et développement de stratégies	Intérêt et motivation
Descripteurs	sentiment de contrôle du jeu	plaisir ressenti	perte momentanée de la notion de temps et de lieu	liberté dans les actions encadrées par les règles du jeu	jugement positif de la qualité du jeu et grande implication dans le jeu
Recherches associées	Anneta, Lamb et Stone (2011), Dondlinger (2007), Garris, Ahlers et Driskell (2002), Kiili (2005), Marfisi et al. (2012), Sanchez (2013) et Van Staalduinen et Freitas (2011)	Amory (2007), Dondlinger (2007), Garris, Ahlers et Driskell (2002), Marfisi et al. (2012), Sanchez (2013) et Van Staalduinen et Freitas (2011)	Amory (2007), Anneta, Lamb et Stone (2011), De Freitas et Oliver (2006), Garris, Ahlers et Driskell (2002), Kiili (2005), Marfisi et al. (2012) et Van Staalduinen et Freitas (2011)	Amory (2007), Anneta, Lamb et Stone (2011), Di Loreto et Gouaich (2010), Dondlinger (2007), Garris, Ahlers et Driskell (2002), Kiili (2005), Marfisi et al. (2012), Sanchez (2013) et Van Staalduinen et Freitas (2011)	Amory (2007), Anneta, Lamb et Stone (2011), Dondlinger (2007), Garris, Ahlers et Driskell (2002), Kiili (2005), Marfisi et al. (2012), Sanchez (2013) et Van Staalduinen et Freitas (2011)

Tableau 2 – Réactions et comportements de l'apprenant.

Un autre point récurrent dans les modèles conceptuels de jeux existants est le *débriefing* (Garris, Ahlers et Driskell, 2002 ; Marfisi et al., 2012 ; Sanchez, 2013). Le débriefing aide l'apprenant à s'approprier les connaissances et les compétences visées par le jeu en analysant les différents événements et résultats ayant été expérimentés dans le jeu. Ainsi, une fois les connaissances validées par l'enseignant, elles deviennent explicites et transférables vers d'autres situations.

1.5.2 *Implications sur le processus de conception et d'évaluation*

1.5.2.1 *Structurer le processus de conception et d'évaluation selon trois dimensions : utilisabilité, utilité et acceptabilité*

Si on suit les modèles et les grilles d'analyse et de conception proposés dans la littérature, la conception d'un bon jeu pour l'apprentissage, relèverait de l'intégration des attributs soulignés par ces modèles dans un artefact informatique. Il s'agit alors de vérifier de façon analytique, l'incorporation d'un certain ensemble d'indicateurs de qualité pendant le processus de conception. Il s'agit également de vérifier de façon empirique certains objectifs de conception telle que l'efficacité des apprentissages, une pratique très répandue dans la littérature (cf. s. 1.4.2).

Le fait de considérer les différents attributs donnés par ces modèles, peut amener à considérer en même temps les trois dimensions *utilisabilité, utilité et acceptabilité*, auxquels ces attributs peuvent être associés (cf. s. 1.4.1), dans le but d'assurer la qualité conceptuelle du jeu. Dans deux de nos communications (Djelil, 2014 ; Djelil, Sanchez et al., 2014), nous avions soutenu l'idée que l'évaluation de jeux pour l'apprentissage peut être structurée sur la base de ces trois dimensions.

L'utilisabilité concerne la possibilité d'utiliser le jeu dans un contexte d'apprentissage et dépend des aspects ergonomiques du jeu-game. En ce sens, l'évaluation de l'utilisa-

bilité pourrait être assurée de façon analytique (en phase de conception) ou empirique (en phase expérimentale), en considérant les aspects liés à l'interface utilisateur du jeu-game, ainsi qu'à certaines réactions de l'apprenant développées lors de ses interactions avec le jeu-game (Tableau 3).

Critère	évaluation analytique	évaluation empirique
Modalités de représentation	- prévoir des modalités attrayantes, pouvant susciter la curiosité et l'intérêt de l'apprenant	- estimer leur appréciation par l'apprenant
Interactions (actions et feedbacks)	- prévoir des interactions intuitives et des feedbacks faciles à interpréter	- estimer la facilité des interactions et leur appréciation par l'apprenant
Divertissement	- vérifier l'incorporation d'éléments ludiques et attrayants susceptible de déclencher un sentiment de divertissement chez l'apprenant	- estimer le ressenti de l'apprenant

Tableau 3 – Évaluation de l'utilisabilité du jeu.

Nous avions également supposé que l'évaluation de l'utilité permettrait de vérifier l'impact de l'introduction du jeu sur l'apprentissage, étant donné qu'elle est centrée sur les aspects didactiques du jeu. Ceci dit, l'évaluation de l'utilité serait liée aux caractéristiques du jeu qui aident à développer chez l'apprenant des réactions impliquant l'acquisition des connaissances et sa progression dans l'apprentissage. De ce fait, l'évaluation analytique permettrait de vérifier, lors de la phase de conception, la prise en comptes des critères de conception liés aux objectifs d'apprentissage et aux caractéristiques du jeu permettant de les atteindre. Par ailleurs, l'évaluation empirique permettrait, lors des phases expérimentales, d'analyser le comportement de l'apprenant afin d'arriver à déduire des indicateurs liés à l'apprentissage, comme la prise de décisions et le développement de stratégies. Un diagnostic des connaissances et compétences doit être également élaboré afin d'évaluer l'utilité du jeu (Tableau 4).

L'acceptabilité d'un jeu dédié à l'apprentissage est liée à son contexte d'exploitation et concerne l'adéquation de ce dernier aux contraintes technologiques et pédagogiques de son utilisation. Elle dépend du jugement de l'apprenant, de l'enseignant et de l'institution porté sur sa valeur. De ce fait, l'évaluation analytique de l'acceptabilité permettrait de vérifier, dès le début de la phase de conception, l'adéquation du contenu du jeu aux contraintes technologiques et organisationnelles du contexte de son utilisation. L'évaluation empirique permettrait de vérifier la capacité du jeu à susciter et à maintenir l'intérêt et la motivation de l'apprenant pour le jeu (Tableau 5).

1.5.2.2 *Distinguer le jeu-play du jeu-game lors du processus de conception et d'évaluation*

L'inconvénient de la démarche conceptuelle présentée précédemment (cf.s. 1.5.2.1) réside dans le fait qu'elle ne dissocie pas le jeu-play du jeu-game. Par conséquent, les frontières entre ces deux parties du jeu deviennent très floues. Cette démarche accentue également l'analyse des attributs associés au jeu-game et induit le concepteur à centrer son attention sur l'artefact. La distinction entre jeu-play et jeu-game permet justement de situer l'apprentissage et d'identifier quelle partie du jeu, artefact ou situation, il faut considérer lorsqu'il s'agit de concevoir et d'évaluer les apprentissages.

Critère	évaluation analytique	évaluation empirique
Objectifs et règles	- vérifier que les objectifs d'apprentissage sont bien définis et qu'ils sont atteignables progressivement en respectant les règles du jeu	
Modalités de représentation	- vérifier la cohérence des représentations des concepts d'apprentissage	
Difficulté progressive et adaptée	- vérifier que les objectifs d'apprentissage sont atteignables de manière progressive	
Échecs et erreurs	- permettre que les échecs et les erreurs soit frivoles	
Interactions (actions et feedbacks)	- prévoir des interactions avec des éléments qui représentent ou décrivent des concepts. Prévoir également des feedbacks constructifs	- peut être vérifié par l'analyse des interactions
Auto-efficacité et contrôle	- prévoir des moments de jeu où l'apprenant peut se sentir en confiance	
Immersion et *flow*	- prévoir un niveau d'immersion adéquat	
Prise de décision et développement de stratégies	- prévoir des mécanismes permettant à l'apprenant de choisir des actions et de les tester	- peut être vérifié par l'analyse du comportement de l'apprenant

Tableau 4 – Évaluation de l'utilité du jeu.

Critère	évaluation analytique	évaluation empirique
Contexte d'apprentissage	- vérifier les conditions techniques et organisationnelles du contexte d'exploitation du jeu	
Débriefing	prévoir un débriefing permettant d'expliciter les connaissances acquises	- vérifier l'état des connaissances de l'apprenant
Intérêt et motivation		- analyse du comportement de l'apprenant et recueil de son jugement

Tableau 5 – Évaluation de l'acceptabilité du jeu.

Par ailleurs, on voit dans ces modèles que lorsqu'il s'agit d'évaluer les apprentissages, ce sont les attributs liés à la situation qui sont considérés plutôt que l'artefact. Cela est en concordance avec la distinction entre le jeu-play et le jeu-game faite par quelques auteurs (Sanchez, Emin-Martinez et Mandran, 2015 ; Shaffer, 2006), qui soutiennent l'idée que l'apprentissage résulte des interactions qui se nouent entre l'apprenant et le jeu-game. C'est durant cette situation que les connaissances de l'apprenant se développent et évoluent en fonction du feedback du jeu-game. L'évaluation des apprentissages peut alors se traduire par l'analyse des usages et des interactions de l'apprenant avec l'artefact et leur relation avec les connaissances développées chez l'apprenant, plutôt que l'artefact lui-même.

1.6 CONCLUSIONS ET IMPLICATIONS

Le travail décrit dans ce premier chapitre vise à comprendre le jeu et sa relation à l'apprentissage. Il possède des implications sur un autre travail de conception et d'évaluation d'une expérience d'apprentissage par le jeu.

Le jeu par ces multiples attributs n'a de lien avec l'apprentissage que lorsqu'il y a interaction de l'apprenant avec sa structure ludique. Cette conclusion nous amène, au même titre que certains chercheurs, à adopter la distinction entre deux notions importantes du jeu et de son utilisation à des fins d'apprentissage, le jeu en tant qu'artefact ou ressource (*jeu-game*) et le jeu en tant que situation résultante de l'interaction de l'apprenant avec cet artefact (*jeu-play*).

La distinction entre jeu-play et jeu-game est non sans conséquences sur le processus de conception et d'évaluation des apprentissages. Cela nous amène à mettre l'accent dans la suite de ce travail sur la conception et l'analyse d'une expérience ludique de l'apprenant. Ainsi, nous considérons que la conception d'un jeu-play ne signifie pas nécessairement l'intégration d'éléments constitutifs de jeux vidéos, tels que des défis et des récompenses dans un artefact informatique. Le jeu-play est vue comme une expérience qui peut être mise en place, notamment par une activité dans laquelle l'apprenant est amené à exercer sa *créativité* et son *imagination*. Aussi, dans ce qui suit, lorsque nous utiliserons le terme *jeu-play*, c'est pour désigner le *jouer* ou la situation plutôt que la ressource (jouet ou jeu-game) employée pour créer cette situation.

Nous adoptons également l'idée que les connaissances de l'apprenant se construisent lors de cette situation de *jeu-play*, lorsque l'apprenant interagit avec l'artefact grâce auquel cette situation est mise en place. Ainsi à la question posée en synthèse de ce premier chapitre (cf. s. 1.5) : "*faut-il considérer la situation ou l'artefact dans l'apprentissage par le jeu ?*", nous répondons qu'il faut considérer à la fois l'artefact et la situation au sein de laquelle des interactions se produisent entre l'apprenant et cet artefact. L'évaluation empirique de l'apprentissage qui est au cœur de notre travail, peut être menée par l'analyse des usages et des interactions de l'apprenant avec l'artefact que nous concevons pour élaborer cette expérience d'apprentissage ludique. Nous considérons également l'importance du sentiment de divertissement et de détermination chez l'apprenant, qui doit émerger des interactions de par le caractère ludique de cette expérience. Il est également important de veiller à ce que certains attributs soient soigneusement intégrés dans le jeu-game (l'artefact informatique), telles que les modalités de représentation qui doivent être conçues de manière fidèle au modèle de connaissances considéré. Dans le chapitre suivant, nous abordons la POO qui est le domaine de connaissances de notre jeu-game.

PUBLICATIONS

Quelques idées et propositions décrites dans ce chapitre sont apparues dans les publications suivantes :

Djelil, Fahima (2014). « Comment évaluer un jeu d'apprentissage en contexte de formation professionnelle ». In : *Rencontres de Jeunes Chercheurs en EIAH*, p. 11–16.

Djelil, Fahima, Eric Sanchez, Benjamin Albouy-Kissi, Jean-Marc Lavest et Adélaïde Albouy-Kissi (2014). « Towards a learning game evaluation methodology in a training context : A literature review ». In : *European Conference on Games Based Learning*. T. 2. Academic Conferences International Limited, p. 676–682.

2

APPRENTISSAGE ET ENSEIGNEMENT DE LA PROGRAMMATION

2.1 INTRODUCTION

L'apprentissage et l'enseignement de la programmation aux débutants s'avèrent difficiles (Bennedsen, 2008 ; Bennedsen, Caspersen et Kölling, 2008 ; Robins, J. Rountree et N. Rountree, 2003). Depuis les années 1970, les questions s'intéressant à l'apprentissage de la programmation, visant à connaître l'origine et les causes des difficultés rencontrées par les apprenants débutants ne cessent d'êtres soulevées (Robins, J. Rountree et N. Rountree, 2003). De même, l'enseignement de la programmation est considéré comme l'un des défis majeurs de la didactique de l'informatique (Mcgettrick et al., 2005). En particulier, l'enseignement des fondamentaux de la programmation au niveau universitaire a fait l'objet de nombreuses discussions impliquant des chercheurs, des enseignants et des didacticiens (Becker, 2001 ; Bruce, 2005 ; Oldham, 2005 ; Woodworth et Dann, 1999). À titre indicatif, l'interrogation de la base de donnés *ACM Digital Library* [1] sur le terme *CS1* [2] renvoie 983 résultats et 97,364 résultats sur l'expression "*introductory programming*" (introduction à la programmation) [recherche réalisée le 23 Mai 2016]. Une tendance plus récente est de mettre l'accent sur l'enseignement et l'apprentissage de la POO, qui est l'un des paradigmes le plus difficile à appréhender par les débutants. Il s'agit pour certains chercheurs de lever des questions autour de nouvelles approches pour améliorer l'enseignement et l'apprentissage des fondamentaux de la POO (Bennedsen et Schulte, 2007 ; Kölling et Rosenberg, 2001 ; Xinogalos, Satratzemi et Dagdilelis, 2006).

Aujourd'hui, justifiées par l'émergence d'une culture numérique, les questions relatives à l'enseignement de l'informatique qui passe obligatoirement par l'initiation à la programmation, s'étendent à tous les niveaux scolaires et sortent du contexte universitaire. La POO fait partie des cours introductifs de la programmation dans les programmes de formations universitaires. Elle sera probablement introduite dans les curriculums d'autres niveaux comme le secondaire dans les prochaines années. Cela nous amène à considérer l'enseignement introductif de la programmation ainsi que les difficultés d'apprentissage des étudiants débutants. C'est exactement l'objectif de ce chapitre qui introduit brièvement, dans un premier temps, les concepts d'algorithmique et de programmation (Section 2.2) avant d'aborder, dans un second temps, les paradigmes de programmation les plus enseignés au niveau universitaire (Section 2.3). L'objectif n'est pas de présenter de manière exhaustive l'ensemble de ces paradigmes, mais de présenter quelques fondements de chacun. Ensuite, nous nous intéressons aux différentes approches d'enseignement des fondamentaux de la programmation au niveau universitaire (Section 2.4), avant de nous concentrer sur le paradigme OO, sa genèse,

1. http://dl.acm.org
2. *CS1* (*Computer Science 1*) désigne la discipline de l'informatique au niveau du premier cycle universitaire dans le programme de formation anglo-saxon.

les difficultés d'apprentissage de ce paradigme et les tendances des recherches menées sur son enseignement (Section 2.5). Nous procèderons à une synthèse visant à tirer des conclusions sur une approche d'enseignement et d'apprentissage des fondamentaux de la POO (Section 2.6). Enfin, nous conclurons sur les implications de ce travail sur le déroulement de la thèse (Section 2.7).

2.2 ALGORITHMIQUE ET PROGRAMMATION

Un *algorithme* est présenté comme une suite d'actions que devra effectuer un automate (ordinateur) pour arriver en un temps fini à un résultat déterminé à un problème (Christophe, 2005 ; Courtin et Kowarski, 1994). La suite d'actions est composée d'opérations élémentaires, dites *instructions*, qui doivent être décrites dans un ordre cohérent (Christophe, 2005). Pour être exécutés sur ordinateur, les algorithmes doivent être traduits dans un *langage de programmation*.

La définition d'un langage de programmation recouvre trois aspects fondamentaux (Granet, 2014). Le premier, appelé *lexical*, définit les symboles (ou caractères) qui servent à la rédaction des programmes et les règles de formation des mots du langage. Par exemple, un entier décimal sera défini comme une suite de chiffres compris entre 0 et 9. Le second, appelé *syntaxique*, est l'ensemble des règles grammaticales qui organisent les mots en phrases. Par exemple, la phrase « `234/54` », formée de deux entiers et d'un opérateur de division, suit la règle grammaticale qui décrit une expression. Le dernier aspect, appelé *sémantique*, étudie la signification des phrases. Il définit les règles qui donnent un sens aux phrases. Par exemple, la phrase « `234/0` », une division par zéro, est invalide d'un point de vue de sa sémantique. L'ensemble des règles lexicales, syntaxiques et sémantiques définit un langage de programmation. La vérification de la conformité lexicale, syntaxique et sémantique d'un programme est assurée automatiquement par des *analyseurs*.

Pour être exécuté sur un ordinateur, un programme rédigé dans un langage de programmation de haut niveau est soit *traduit* ou *interprété* en un code compréhensible par la machine :

— le programme écrit dans un langage de haut niveau, dit *source*, est *traduit* en un programme *sémantiquement* équivalent écrit dans le langage machine de l'ordinateur. Cette traduction est faite par un *compilateur*. Un compilateur possède au moins quatre phases : trois phases d'analyse (lexicale, syntaxique et sémantique) et une phase de production de code machine. Le compilateur ne produit le code machine que si le programme source respecte les règles du langage, sinon il devra signaler les erreurs au moyen de messages précis.

— l'*interprétation* d'un programme est réalisée en passant par une première phase de traduction du langage de haut niveau vers un langage intermédiaire de plus bas niveau. Cette phase de traduction comporte les mêmes phases d'analyse qu'un compilateur. L'interprétation est alors faite sur le langage intermédiaire. L'objectif de cette méthode est d'assurer la portabilité des programmes.

Le développement et l'analyse d'erreurs (ou *bugs*) d'un programme peut se faire à l'aide d'un *débogueur*, qui permet d'exécuter un programme pas à pas, d'afficher la valeur des variables lors du déroulement de l'exécution du programme et de mettre

des points d'arrêt à des niveaux différents du programme. Ces opérations constituent le *débogage* d'un programme.

Le but de tout programme est de calculer et retourner des résultats valides et fiables. L'activité de programmation exige des méthodes rigoureuses, si les objectifs de justesse et fiabilité veulent être atteints (Granet, 2014). Jusque dans les années 1960, la structuration des programmes n'était pas un souci majeur. C'est à partir des années 1970, face à des coûts de développement des logiciels croissants, que l'intérêt pour la structuration des programmes s'est accrue. La manière dont un programme est construit constitue un *style* de programmation ou un *paradigme* de programmation. Nous décrivons ce concept dans la section suivante.

2.3 PARADIGMES DE PROGRAMMATION

2.3.1 *Paradigme impératif et paradigme fonctionnel*

Le paradigme impératif appelé aussi paradigme procédural s'appuie sur le principe d'un *état mémoire implicite* qui peut être modifié par l'application de *commandes* (Hudak, 1989). Les langages de programmation impérative sont fondés sur le principe de *séquencement* de commandes permettant de contrôler de façon déterministe et précise un état mémoire implicite. Un exemple de ce type de programmation, est une instruction (ou commande) d'affectation dont l'effet est d'altérer un espace de stockage désigné de manière implicite par une variable. Étant donné qu'une variable renvoie à un emplacement mémoire, à la notion de variable est attaché la notion de *type* qui permet de délimiter la taille de l'emplacement mémoire. Les langages impératifs emploient des délimiteurs de commandes, des instructions de contrôle conditionnelles et itératives dans la manipulation d'un état mémoire. L'exemple suivant montre le principe de la mémoire implicite, il donne un programme qui calcule la factorielle d'un nombre `x` :

```
n:=x;
a:=1;
while n>0 do
   begin
       a:=a*n;
       n:=n-1;
   end;
```

Après exécution de ce programme, la variable `a` dans l'espace de stockage implicite va avoir la valeur souhaitée. Ce principe n'est, en revanche, pas présent dans le paradigme de programmation *fonctionnel*. Les langages de programmation fonctionnelle n'ont pas d'état implicite, de ce fait l'accent est entièrement mis sur la programmation avec des *expressions* ou *termes* (Hudak, 1989). Leur modèle computationnel sous-jacent est fondé sur la *fonction*. Il s'agit d'évaluer des fonctions mathématiques pour réaliser des calculs.

Le paradigme fonctionnel est apparu depuis de nombreuses années. Il tient son origine du langage LISP en 1958 (Sturm, 2011). L'un des concepts importants de ce paradigme est celui de la *transparence référentielle* des fonctions qui sous-entend que la valeur de retour d'une fonction dépend uniquement de ses paramètres d'entrée et non pas de l'état du programme, comme c'est le cas dans le paradigme impératif. Une autre

propriété importante est la notion de *récursivité* des fonctions. Une fonction récursive est une fonction qui fait appel à elle-même dans sa propre définition. Les langages de programmation fonctionnelle définissent la récursivité de manière plus lisible que les langages de programmation impérative, qui utilisent des structures de contrôle itératives pour exprimer la récursivité. À titre d'exemple, la factorielle d'un nombre x peut être calculée par le langage fonctionnel Ocaml comme suit :

```
let rec fact n=
    if n=0 then 1 else n * fact (n-1)
```

À noter que dans ce programme, la condition est une expression et non pas une commande, de ce fait elle dénote une valeur. La valeur du programme est la valeur de la factorielle calculée, plutôt qu'une valeur stockée dans un emplacement implicite.

Un autre mécanisme propre aux langages fonctionnels est l'usage des fonctions dites *d'ordre supérieur*, qui sont des fonctions qui prennent en arguments d'autres fonctions dans leurs définitions et renvoient d'autres fonctions comme résultats.

2.3.2 *Paradigme Orienté-Objet*

Le paradigme OO définit les principes de la POO qui sont fondés sur la dualité *classe/instance* ou *classe/objet*. Le principe classe/instance de la POO étend le principe *type/variable* de la programmation procédurale qui a précédé l'OO. Tout comme un type établit les propriétés d'une variable, une *classe* est un moule, un gabarit pour l'*instance* (l'*objet*) (Barbier, 2009). À la notion de classe est liée la notion de *constructeurs de classes*, qui sont des fonctions définissant les termes et conditions d'allocations des objets en mémoire et leur initialisation. Un constructeur est par définition une fonction n'ayant pas de type de retour et ayant obligatoirement pour nom le nom de sa classe.

L'*abstraction* est un autre principe fondamental de ce paradigme. Il s'agit de la capacité de représenter quelque chose de façon incomplète (Barbier, 2009). La représentation est délibérément réductrice de la chose, pour s'affranchir de la complexité inhérente à cette chose. Construire une classe en OO est par définition produire une abstraction. L'une des propriétés majeures de ce paradigme, est le confinement des traitements et des données à l'intérieur des classes. L'ensemble des données (*attributs*) et des traitements (*méthodes*) constituent les propriétés d'une classe. L'abstraction est instrumentée au moyen d'un autre concept, dit *encapsulation*. Une partie de la classe est délibérément masquée (ou *encapsulée*) pour rendre l'accès à la classe plus aisé de l'extérieur. L'intérieur invisible du coté utilisateur rend la classe moins complexe à utiliser.

L'*héritage* et le *polymorphisme* sont d'autres concepts essentiels à ce paradigme. Ce sont des mécanismes d'appariement des classes en vue de rationaliser l'organisation des bibliothèques de code et des programmes. L'héritage permet d'abord d'apparier des structures de données, c'est *l'héritage de structure* (ou *héritage structurel*) : une classe *héritant* (ou *dérivant*) d'une autre dispose pour ses instances des propriétés de sa classe ascendante directe ou plus, si cette dernière a elle-même une ascendante (Barbier, 2009). Quand l'héritage concerne les opérations à l'intérieur d'une classe, il est dit *héritage de comportement* (ou *héritage comportemental*). Ce type d'héritage engendre le *polymorphisme* qui n'a pas de corrélation avec l'héritage de structure. Le polymorphisme permet de

faire la distinction entre le type déclaré d'un objet et la façon dont ce dernier est créé puis connu en mémoire. L'information se trouvant en mémoire pour connaître à tout moment le type à l'exécution est soit le type déclaré soit celui d'un sous-type. Le polymorphisme exploite cette information présente en mémoire pour exécuter la bonne opération associé à l'objet à un moment donné.

Un autre concept de base important de l'OO est la *classe abstraite*. Une classe quelconque étant une abstraction de la réalité, une classe abstraite ne signifie pas la même chose et possède un sens technique différent. Une classe abstraite est par définition une classe à partir de laquelle il n'est pas possible de créer des objets. Elle s'utilise et n'a d'intérêt qu'avec l'héritage et le polymorphisme (Barbier, 2009). A la notion de classe abstraite est associée la notion de *fonction abstraite* (ou *fonction virtuelle*), qui est rendue concrète dans les classes héritant de la classe abstraite. Le principe de la classe abstraite combiné au polymorphisme permet une meilleure organisation mémoire d'un programme.

La *généricité* est un autre concept important qui est instrumenté par la notion de *fonctions génériques*. Le principe de la généricité est de permettre à un programme d'écrire du code manipulant un type de données anonyme pour que ce type soit ensuite remplaçable par un autre type quelconque (Barbier, 2009).

Nous allons maintenant étayer les approches d'enseignement introductif de la programmation, qui sont répandues en pratique et classifiées par l'ACM et l'IEEE.

2.4 Approches d'enseignement introductif de la programmation

Les six approches d'enseignement introductif de la programmation décrites dans cette section sont tirées du rapport CS2001 (*Computing curricula 2001*)[3] publié par l'ACM et l'IEEE (Engel et E. Roberts, 2001). Cette classification a été réalisée sur la base des pratiques d'enseignement les plus répandues en formation universitaire. Ce rapport décrit les pratiques de chaque approche en soulignant ses avantages et inconvénients, impliqués par le fait que chaque approche se concentre sur un style de programmation particulier au tout début des enseignements.

Impératif-d'abord (*imperative-first*). Cette approche est souvent réalisée de deux façons. Elle peut couvrir les trois enseignements : *les fondamentaux de la programmation, le paradigme OO, les structures de données et les algorithmes*, comme elle peut couvrir les deux enseignements : *introduction à la programmation et abstraction de données*. Cette approche introduit, dans un premier temps, la programmation suivant le style impératif et se concentre, dans un second temps, sur le paradigme OO. Ce qui la différencie de l'approche *Objets-D'abord*, est l'accentuation des premiers enseignements et l'ordre dans lequel ils sont dispensés. Même si les premiers enseignements se font dans un langage OO, le premier cours se concentre sur les aspects impératifs du langage : expressions, structures de contrôles, procédures et fonctions et autres éléments du paradigme procédural classique. Les méthodes de conception OO sont introduites dans les cours suivants.

3. Le rapport CS2001 a été révisé en 2008 puis en 2013, mais les approches d'enseignement de la programmation restent inchangées. http://www.acm.org/education/curricula-recommendations

Cette approche implique que les étudiants soient moins confrontés aux techniques de la POO, que s'ils suivaient une formation ayant pour approche *Objets-D'abord*. Étant donné la place centrale qu'occupe la POO dans la formation et la difficulté que les étudiants rencontrent lorsqu'ils apprennent à programmer dans le paradigme OO, le fait que l'introduction de cet enseignement se fasse tardivement est considérée comme une faiblesse de cette approche. En même temps, les étudiants ont besoin d'être exposés au paradigme impératif de la programmation, qui reste très largement utilisé et fait partie intégrante de n'importe quel langage de POO. Les opinions sont alors divisées entre ces deux approches. Certains argumentent que les étudiants ayant appris le modèle impératif en premier, éprouvent plus de difficultés avec l'approche OO. Tandis que d'autres, au contraire, pensent que les étudiants qui sont habitués à travailler dans un langage OO, s'irriteront à l'idée d'apprendre à travailler sans les propriétés qui rendent la POO si performante. Dans tous les cas, les institutions qui adoptent l'approche impérative en premier, auront besoin d'inclure les bases de la conception OO à un niveau intermédiaire.

OBJETS-D'ABORD (*object-first*). Cette approche accorde une grande importance aux fondamentaux de la programmation et de conception OO au tout début de la formation. Elle peut couvrir les trois enseignements : *introduction à la POO*, *Objets et abstraction de données*, *algorithmes et structures de données*, comme elle peut couvrir les enseignements : *POO*, *conception et méthode OO*. Les premiers enseignements abordent rapidement les notions d'objets et d'héritage, afin de confronter les étudiants très tôt à ces concepts. Après avoir fait l'expérience de ces notions dans un contexte de programmes interactifs simples, les enseignements passent alors à l'introduction des structures de contrôles classiques, mais toujours en se concentrant sur la conception OO. Les enseignements suivants vont ensuite introduire les algorithmes, les structures de données fondamentales et le génie logiciel dans plus de détails.

Le principal avantage de cette approche est la confrontation précoce à la POO, qui prend de plus en plus d'importance, dans les milieux académiques et industriels. En même temps, cette approche peut présenter des inconvénients, liés à la complexité des langages de programmation utilisés tels que C++ et Java. Les détails de la POO peuvent surcharger les étudiants débutants, même si les enseignants introduisent avec beaucoup de soins ces notions.

FONCTIONNEL-D'ABORD (*functional-first*). Cette approche se caractérise par l'utilisation d'un langage de programmation fonctionnelle dès les premières leçons.

Comparativement aux autres approches, elle présente essentiellement les avantages suivants : elle limite les effets de la diversité des parcours des étudiants débutants qui arrivent à l'université, étant donné que les langages fonctionnels sont peu utilisés en dehors de l'université. Les langages fonctionnels présentent une syntaxe minimaliste, ce qui implique que les cours peuvent se concentrer sur les concepts fondamentaux. Enfin, plusieurs notions importantes telles que, la récursivité, les listes chaînées et les fonctions peuvent être abordées très tôt de façon naturelle. Néanmoins, les étudiants peuvent être sceptiques à l'idée d'apprendre un langage qui n'est pas dominant. Pour les étudiants ne connaissant pas l'informatique, ayant pour objectif de découvrir cette

discipline avant de se décider d'aller plus loin, un cours introductif utilisant un langage fonctionnel pourrait être dissuasif. Un autre danger de cette approche est le fait qu'elle requiert que les étudiants pensent avec plus d'abstraction dès le début, que s'ils utilisaient un langage de programmation plus classique. Inciter les étudiants à penser de la sorte est certainement valable et mérite de faire partie de la formation, mais le placer au tout début de leur formation pourrait dissuader les étudiant les moins expérimentés.

ÉTENDUE-D'ABORD (*breadth-first*). Cette approche est une alternative aux autres approches qui orientent les enseignements sur la programmation. Elle vise à offrir une vue plus globale et étendue de l'informatique et voit dans les autres approches le risque d'offrir aux étudiants une vue limitée de cette discipline.

L'avantage de cette approche, est qu'elle offre aux étudiants une appréciation rapide de l'informatique, leur permettant de décider plus facilement s'il s'agit d'un domaine qu'ils souhaiteraient étudier en profondeur. Son inconvénient principal, est qu'elle allonge les enseignements introductifs de l'informatique. Une autre façon d'exécuter cette approche, consiste à prévoir une revue du champs disciplinaire à la suite des cours introductifs de la programmation. Cela permet aux étudiants d'acquérir des compétences en programmation, tout en ayant la possibilité d'apprécier très tôt les autres matières de la discipline.

ALGORITHMIQUE-D'ABORD (*algorithms-first*). Dans cette approche les concepts de base de la programmation sont introduits au moyen d'un pseudocode à la place d'un langage exécutable. Les étudiants résonnent sur des algorithmes qu'ils construisent à la main. De cette manière, ils se soucient moins des détails de syntaxe indispensables à l'exécution d'un programme. Cela leur permet de travailler avec plusieurs structures de contrôles et structures de données sans être confrontés aux particularités des langages de programmation classiques. Une fois que les étudiants ont gagné une solide compréhension des concepts algorithmiques du pseudocode, ils peuvent passer à un langage de programmation plus classique. Les concepts fondamentaux de l'algorithmique font partie intégrante de tout langage de programmation. Le fait que les étudiants soient confrontés très tôt à ces concepts rend le passage à d'autres langages de programmation plus aisé. De ce fait, les enseignements peuvent se consacrer à l'optimisation et au débogage de programmes.

Cette approche présente, néanmoins, plusieurs inconvénients. Le premier est le fait que les étudiants n'acquièrent pas des compétences pratiques en programmation, c'est pourquoi il est recommandé de compléter cette approche avec des activités qui exposent les étudiants à de vraies applications afin de leur offrir une expérience appliquée de l'algorithmique. Le fait que cette approche repose sur du pseudocode ne fait pas confronter les étudiants au besoin de vérifier l'exécution de leurs programmes. Or, la compilation et l'exécution de programmes est une compétence essentielle qu'il faut acquérir très tôt.

MATÉRIEL-D'ABORD (*hardware-first*). Cette approche enseigne les bases de l'informatique en commençant au niveau de la machine et en évoluant vers des concepts plus abstraits. Les enseignements commencent en introduisant les circuits numériques qui constituent les registres et les unités arithmétiques, en passant par la machine de

Von Neumann[4]. Une fois ces notions sont établies, les enseignements introduisent la programmation dans un langage de haut niveau. Cette approche couvre en général les enseignements : *introduction à l'ordinateur et techniques de POO*.

Cette approche peut convenir aux étudiants qui préfèrent un apprentissage détaillé de l'informatique, mais peut être dissuasive pour les étudiants qui préfèrent ne pas aller au-delà des aspects de l'implémentation. Elle est également en contradiction avec les tendances des machines virtuelles actuelles qui séparent le processus de programmation de la partie hardware. Cette approche reste néanmoins valable dans un programme de formation en génie informatique, où une confrontation précoce au hardware est essentielle.

La section suivante se consacre à la POO et s'intéresse aux recherches menées sur son enseignement et son apprentissage. Le but étant, d'une part, de comprendre l'origine des difficultés que les étudiants rencontrent lors des enseignements introductifs, et d'autre part, de connaître les approches ayant été mises en œuvre pour palier ces difficultés.

2.5 Programmation orientée-objet (POO)

2.5.1 *Genèse de la POO*

La POO est née avec le langage *Simula*, qui a été introduit par Ole Johan Dahl et Kristen Nygaard lors de la conférence "IFIP TC 2 Working Conference on Simulation Languages" à Oslo en 1967 (Sklenar, 1997). Tous les langages OO qui ont été conçus plus tard se basent sur les principes de la POO introduits pour la première fois dans le langage Simula, telles que les notions de classes, d'instances et d'héritage. Ci-dessous, un exemple de déclaration et d'instanciation d'une classe Rectangle ayant deux attributs (`Area et Perimeter`) et deux méthodes (`Update et IsSquare`) avec Simula (Sklenar, 1997) :

```
Class Rectangle (Width, Height); Real Width, Height
 Begin //debut de delaration de la classe Rectangle
    Real Area, Perimeter;
    Procedure Update;
    Begin
      Area := Width * Height;
      Perimeter := 2*(Width + Height)
    End of Update;
    Boolean Procedure IsSquare;
      IsSquare := Width=Height;
    Update;
    OutText("Rectangle created: "); OutFix(Width,2,6);
    OutFix(Height,2,6); OutImage
 End of Rectangle; //Fin de delaration de la classe Rectangle
```

4. La machine de Von Neumann porte le nom de son concepteur, le mathématicien John von Neumann qui en 1946, a élaboré la première description d'un ordinateur dont le programme est stocké dans une mémoire.

```
 <...>
Ref(Rectangle) R; //declaration d'une reference d'objet R, instance de la classe
    Rectangle
R := New Rectangle(50, 40);
```

Les travaux sur la POO ont été poursuivis par Alan Kay qui ressort deux idées centrales liées à ce style de programmation. Il s'agit, d'une part, de trouver un schéma modulaire d'un système complexe en décrivant ses propriétés tout en occultant ses détails, et d'autre part, de trouver une manière plus flexible d'effectuer des assignations (Kay, 1996). Influencé par les travaux de (Papert, 1980) sur le langage LOGO (cf.s. 3.2.2.1), les potentialités du langage Simula et l'avènement des ordinateurs personnels, Alan Kay introduit le langage *SMALLTALK* dès 1972, dans le cadre d'un groupe de recherche à Xerox Parc [5]. SMALLTALK apportait déjà un grand nombre de concepts OO mettant en évidence six idées principales (Kay, 1996) : 1) tout est objet, 2) les objets communiquent par envoie de messages, 3) les objets disposent d'une mémoire propre, 4) chaque objet est une instance d'une classe, 5) une classe décrit le comportement commun de ses instances (objets dans un programme), 6) les expressions sont évaluées de sorte à ce que le premier objet dans une expression reçoive en message le reste de l'expression. SMALLTALK implémente également les mécanismes d'héritage et de polymorphisme (Sharp, 1996).

En 1979, Bjarne Stroustrup créa le langage *C with classes* (C avec des classes), qu'il renomma ensuite C++ en 1984 (Stroustrup, 1996). Le langage C++ a été conçu sur la base des langages Simula et C. Le but étant de tirer profit des fonctionnalités offertes, d'un coté, par Simula dans l'organisation de programmes, et d'un autre coté, de l'efficience et de la flexibilité du C dans la programmation de systèmes (Stroustrup, 1996). Simula a fournit les fondements de base aux mécanismes d'abstraction des données du C++ qui a ensuite évolué depuis la moitié des années 1980, sous l'influence de plusieurs autres langages de programmation et la contribution de plusieurs autres personnes (Stroustrup, 2013). C++ a été standardisé par l'ISO [6] en 1990, il est devenu l'un des langages de programmation les plus populaires. Aujourd'hui le langage continue à s'étendre par l'ajout de nouvelle bibliothèques. Sa dernière mise à jour dont la documentation est finalisée a été approuvée en 2011. En 2014, une légère extension du langage a été réalisée (C++14) et de nouvelles mises à jour sont prévues pour les années à venir [7].

L'un des langages OO qui ont marqué les débuts de la POO est le langage *Common Lisp*, dont la spécification constituait le *Common Lisp Object System* fondé sur les concepts de fonctions génériques, d'héritage multiple et d'opérateurs (DeMichiel et Gabriel, 1987). *Common Lisp* a été conçu dans le cadre d'une collaboration entre deux groupes

5. Xerox Parc est l'appellation ancienne du Palo Alto Research Center (abrégé PARC), qui est un centre de recherches en informatique situé à Palo Alto en Californie (États-Unis).

6. l'ISO est un organisme de normalisation international dans les domaines industriels et commerciaux.

7. La documentation des différentes extensions du langage C++ sont disponibles sur http://fr.cppreference.com et http://www.cplusplus.com

de recherche de Symbolics [8] et de Xerox PARC en 1986. Il fut le premier langage Objet standardisé par l'ANSI [9] (DeMichiel et Gabriel, 1987).

À la fin des années 1980, l'OO a suscité l'intérêt d'importants industriels et plusieurs langages de programmation Objet ont été conçus (A. P. Black, 2013). Snyder (1991) a publié une revue de littérature décrivant et classifiant les concepts d'un système OO implémentés dans les langages de programmation connus à cette époque, tels que *SMALLTALK, C++ et Common Lisp*. Snyder (1991) décrit un système OO comme un système contenant des entités (les objets) qui ont un rôle visible par les services qu'ils fournissent. La natures des clients et des objets dépend des particularités du système objet. Un client peut être une personne ou un programme, tant dis qu'un service est tout activité qui peut être réalisée sur la demande d'un client. Les concepts ressortis sont décrits comme suit (Snyder, 1991) :

— un objet donne corps à une abstraction se caractérisant par des services : un objet contient une abstraction qui est significative pour ses clients, il n'est pas uniquement vu comme une structure de données, mais comme un moyen de représentation d'informations. L'abstraction contenue dans l'objet constitue un ensemble de services qui peuvent être invoqués par des clients. Un service peut modifier des données et provoquer des effets observables sur d'autres objets. Les données qui peuvent être modifiées dans l'objet constituent l'état de ce dernier.
— des clients font des requêtes de services : les clients respectent l'abstraction des données à l'intérieur des objets. Au lieu d'accéder directement aux données, les clients émettent des requêtes sur les services fournis associés aux objets. Le client s'intéresse uniquement au retour de la requête et ne se soucie pas des détails de son exécution.
— de nouveaux objets peuvent être créés : des clients peuvent émettre des requêtes de création d'objets.
— les opérations peuvent être génériques : un service peut être implémenté différemment dans différents objets. Un client peut émettre des requêtes uniformes sur un service et une implémentation appropriée est sélectionnée pour chaque requête.
— les objets peuvent être classés en termes de leurs services : les services associés aux objets peuvent être regroupés dans des interfaces. Une interface spécifie comment un objet peut être utilisé par un client, en décrivant un ensemble de requêtes potentielles qui identifient l'objet. Cette spécification peut inclure des informations tels que les paramètres de requêtes et les résultats possibles de ces requêtes. Il s'agit de l'ensemble d'opérations spécifiées dans une classe dans le langage C++, par exemple.
— les objets peuvent partager des implémentations : l'implémentation de services associés à un objet spécifie à la fois la forme de données et le code utilisé pour opérer les services. Les objets ayant une implémentation commune, possèdent des formats de données identiques et partagent un code exécutable. Il s'agit du concept de classe dans les langages SMALLTALK et C++.

8. Symbolics est une entreprise informatique américaine fondée en 1979 dont l'objectif était de commercialiser les machines Lisp.

9. l'American National Standards Institute (ANSI) est un organisme de supervisons de normes de divers produits et systèmes aux États-Unis

Depuis les débuts des années 1990, la POO n'a cessé d'évoluer à la fois dans les aspects théoriques et pratiques. Plusieurs autres langages de programmation sont apparus et devenus très populaires. C'est le cas par exemple des langages Java [10] et C# [11].

2.5.2 *Recherches menées sur les difficultés d'apprentissage des fondamentaux de la POO*

L'apprentissage et l'enseignement des fondamentaux de la POO s'avèrent difficiles. Depuis plus d'une vingtaine d'années, plusieurs auteurs se sont intéressés à comprendre la nature et les causes des difficultés rencontrées par les étudiants lors des enseignements introductifs de la POO (Bashiru et Aderogba Joseph, 2015 ; Biju, 2013 ; Carroll, Rosson et Singley, 1993 ; Kölling, 2003 ; Robins, J. Rountree et N. Rountree, 2003).

Carroll, Rosson et Singley (1993) parlent d'un problème majeur chez les débutants qui est lié à la compréhension des *relations entre objets* et la *création d'objets collaboratifs*.

Nguyen et B. Wong (2001) identifient une difficulté majeure qui est la pensée abstraite, indispensable dans l'apprentissage de la POO. Ils soutiennent l'approche *Objets-D'abord* (cf. s. 2.4) dans l'enseignement des fondamentaux de la POO et de l'abstraction.

Dans leur enquête conduite au Royaume-Uni auprès d'étudiants de deuxième année universitaire et d'enseignants de programmation, Milne et Rowe (2002) identifient les concepts suivants : *fonctions virtuelles, polymorphisme, fonctions génériques, héritage, classe, objet, encapsulation* et la notion de *constructeur/destructeur*. Ils concluent que les difficultés au niveau de ces concepts sont dues au fait que les étudiants ne parviennent pas à comprendre ce qui se produit en mémoire d'ordinateur lors de l'exécution des programmes. Ils expliquent qu'un étudiant continuera à avoir des problèmes de compréhension, tant qu'il n'a pas acquis un modèle mental du fonctionnement de programmes en mémoire (comment un programme est sauvegardé en mémoire et comment les objets sont liés entre eux). Ceci les amènent à justifier leur intérêt pour les outils de visualisation de programmes (Milne et Rowe, 2002).

Certains auteurs soutiennent l'idée que la difficulté d'apprentissage de la POO rencontrée par les débutants n'est pas due à une quelconque complexité inhérente au paradigme OO, mais à d'autres facteurs externes. C'est les cas de Kölling et Rosenberg (2002) qui listent un certain nombre de facteurs incluant :

- un manque d'expérience dans l'enseignement introductif de la POO.
- la non familiarité des enseignants avec le paradigme OO.
- un manque d'expérience de plusieurs auteurs de manuels de cours de l'OO.
- l'inadaptation de certains supports logiciels d'enseignement au paradigme OO. Plusieurs environnements de développements sont des variantes d'environnements de programmation procédurale et n'arrivent pas à montrer l'intérêt de l'OO.
- les enseignements introductifs évoluent d'une approche centrée sur l'algorithmique vers une approche centrée sur la conception logicielle.
- l'évolution technologique influence également le contenu des enseignements introductifs.

Kölling (2003) accuse également les manuels d'enseignement. Dans son enquête réalisée sur 39 manuels de cours d'introduction à la programmation, il souligne leur

10. http ://www.java.com
11. https ://www.microsoft.com/net/default.aspx

inadaptation à l'enseignement : « *les manuels d'enseignement sont structurés selon les spécificités du langage de programmation considéré et non pas sur les techniques de programmation qui devraient être enseignées aux étudiants* » (Kölling, 2003).

Biju (2013) identifie suite à une enquête auprès d'étudiants débutants, des difficultés de compréhension au niveau des concepts de base tels que *l'encapsulation, la classe, l'objet et l'utilisation de constructeurs*. Il exprime suite à cette étude, son intérêt pour les outils de programmation visuelle (cf. s. 3.3) et évoque son intention d'utiliser l'environnement Alice (cf.s.3.3.3.2) dans l'introduction des concepts de base de la POO aux débutants.

Dans leur enquête réalisée auprès d'étudiants de deuxième et de troisième année universitaire au Nigeria, Bashiru et Aderogba Joseph (2015) rapportent que 146 étudiants sur 155 (94%) considèrent que la POO est difficile à apprendre. Les principales difficultés identifiées sont les suivantes :

- ne pas comprendre ce qui se passe exactement à l'intérieur d'un ordinateur quand un programme s'exécute.
- utilisation d'environnements de développement professionnels.
- ne pas comprendre les opérations en mémoire.
- test et débogage de programmes, lecture de code préexistant et détection d'erreurs de logique.
- détection d'erreurs dans son propre programme.
- conception de programmes et développement de solutions pour la résolution de problèmes.
- ne pas comprendre les concepts de base de la POO.
- ne pas comprendre la relation entre l'objet et la classe.

Cette enquête a également réuni les attentes des étudiants et leurs jugements quant à ce qui pourrait améliorer leur apprentissage. Les auteurs ont retenu principalement les solutions suivantes (Bashiru et Aderogba Joseph, 2015) :

- une pédagogie plus efficace pour l'enseignement et l'apprentissage de la POO.
- l'enseignement devrait se concentrer sur la décomposition des programmes et la conception.
- utilisation d'exercices pratiques proches d'expériences réelles.
- les cours devraient être plus visuels et plus interactifs.
- utilisation d'Environnement de Développement Intégré (EDI) modernes.

Par ailleurs, il reste toujours difficile d'identifier les vraies causes des difficultés d'apprentissage chez les débutants. Bashiru et Aderogba Joseph (2015) rapportent que ces difficultés sont à la fois dues aux pratiques d'enseignement mais aussi aux attitudes des étudiants. Ils listent les facteurs suivants :

- non efficacité des méthodes d'enseignement.
- lors de l'écriture ou de la lecture de code exécutable, les étudiants ne parviennent pas à mentalement lier ce qui se passe dans l'ordinateur concernant les changements qui s'effectuent en mémoire et l'état des objets sur lesquels agit le programme.
- manque d'outils et de ressources d'enseignement et d'aide à l'apprentissage.
- complexité des environnements de développement utilisés.
- précipitation sur le codage sans avoir réfléchi à la solution ni avoir en tête la structure du programme.

— manque d'intérêt des étudiants pour l'apprentissage de la POO.

2.5.3 *Recherches menées sur l'enseignement des fondamentaux de la POO*

Dans le milieu des années 1990, l'approche *Objets-D'abord* (cf. s. 2.4) a beaucoup retenu l'attention des chercheurs en enseignement introductif de la POO (Bennedsen, 2008). À titre indicatif, l'interrogation de la base de donnés *ACM Digital Library* sur l'expression *Object-First approach* (*approche Objets-D'abord*) retourne 159.196 résultats [recherche réalisée le 23 Mai 2016]. Cette approche s'est développée dans le but de pallier la difficulté d'apprentissage rencontrée par les étudiants débutants. De ce fait, la recherche sur l'enseignement introductif de la POO a beaucoup été impactée par l'approche *Objets-D'abord* (Joel Adams et Frens, 2003 ; Becker, 2001 ; Bennedsen et Schulte, 2007 ; Buck et Stucki, 2000 ; Cooper, Dann et Pausch, 2003 ; Kölling, 2003 ; Kölling et Rosenberg, 2001, 2002 ; Woodworth et Dann, 1999). Une revue de la littérature plus étendue est donnée dans la thèse de Bennedsen (2008) : « *Teaching and learning introductory programming. A model-based approach*", révélant plusieurs tendances dans les approches adoptées par les auteurs ».

Afin de comprendre comment l'approche "*Objets-D'abord*" est mise en application, Bennedsen et Schulte (2007) ont réalisé une analyse du contenu de plus de 200 réponses d'enseignants à travers le monde. Trois catégories du concept ont été identifiées :

utilisation d'objets : au tout début des enseignements, les étudiants utilisent des objets préalablement implémentés. Dès que les étudiants ont compris le concept d'*objet*, ils passent à la définition de *classes* par eux même. L'accent est mis d'avantage sur l'*usage avant l'implémentation*.

création de classes : l'étudiant *définit et implémente* des classes et crée des *instances* des classes définies. L'accent est mis sur la *partie concrète et créative de la programmation*.

concepts : les étudiants apprennent des *principes et des idées générales* du paradigme OO, en se concentrant sur la création de modèles OO en général et pas seulement sur la programmation. L'accent est mis sur *les aspects conceptuels* de l'OO.

Plusieurs auteurs ont décrit leurs expériences d'enseignement introductif de la POO, en adoptant une approche plus au moins similaire à cette description. Ainsi, Woodworth et Dann (1999) proposent une approche fondée sur la programmation évènementielle, dans laquelle ils préconisent dans un premiers temps, d'attirer l'attention des étudiants sur les principes de résolution de problèmes et les structures de contrôles standards dans des applications consoles en utilisant le langage de programmation C++. Puis, guider très tôt les étudiants, dans la conception et l'utilisation de leurs propres abstractions. Dans cette seconde phase, les étudiants sont amenés à créer des classes en modélisant les propriétés des objets du problème. Enfin, souligner dans une troisième phase, l'existence de deux types de fonctions : les *méthodes de classes* et les méthodes de la Standard Template Library (STL) (*bibliothèque standard*) du C++. Des algorithmes sont utilisés pour développer les fonctions qui modélisent le comportement des objets du problème (les méthodes de classe). L'accent est porté sur la conception avant l'implémentation.

Buck et Stucki (2000) proposent de structurer un cours introductif de la POO de façon à permettre aux étudiants de débuter en modifiant un code préalablement implémenté, puis de concevoir des parties du système. Ils soutiennent une approche où les détails d'un langage de programmation doivent être introduits de façon progressive selon les besoins.

Kölling et Rosenberg (2001) ont proposé un ensemble de directives pour l'enseignement introductif de la POO, qu'ils ont mis en application en utilisant l'environnement BlueJ et le langage Java. Leur démarche met l'accent sur les objets dès le début, au lieu de la programmation impérative. Un étudiant doit être en mesure de créer des objets à partir de classes préalables, de comprendre la notion de l'état d'un objet et d'invoquer des méthodes sur des objets avec ou sans paramètres. Ils recommandent, dans un second lieu, que les étudiants débutent en modifiant des programmes de code existants. Ils expliquent que demander aux étudiants de commencer à programmer par eux même est une erreur d'enseignement, car c'est une activité qu'ils ne peuvent pas comprendre à ce stade. Les étudiants doivent également lire du code bien construit, afin de le reproduire et d'acquérir des bonnes pratiques de programmation. Pour cela, ils recommandent d'utiliser des projets montrant des exemples d'applications constituées de plusieurs classes contenant plusieurs méthodes. Cela permet aussi de montrer l'intérêt de ce paradigme. Deux autres directives recommandent de ne pas utiliser la fonction "*main*" [12] et le programme "*Hello world*" [13]. Kölling et Rosenberg (2001) argumentent que la fonction *main* n'est pas en lien avec le paradigme OO, son intérêt étant de lancer l'exécution d'une application. Quant au programme "*Hello world*", il contient du code qui n'est pas OO. Il est également recommandé de montrer aux étudiants la structure d'un programme comprenant les relations hiérarchiques entre classes. Enfin, une dernière directive concerne l'interface utilisateur de l'application exemple par laquelle un étudiant débute. Kölling et Rosenberg (2001) préconisent que les étudiants puissent expérimenter les entrées/sorties d'un programme via une interface facile, qui permet de visualiser l'exécution de méthodes et le passage de paramètres. L'ensemble de ces directives ont été implémentées dans l'environnement Greenfoot (cf. s. 3.3.3.3) (Kölling, 2010a).

Becker (2001) propose de structurer un cours introductif de la POO selon cinq phases en utilisant l'environnement Karel (cf. s.3.3.3.1) : la première phase est consacrée à l'instanciation et l'utilisation d'objets, dans laquelle les étudiants sont amenés à créer des objets et invoquer des méthodes sur les objets. La seconde phase est dédiée à l'héritage, où les étudiants sont amenés à étendre des classes existantes afin d'obtenir de nouveaux objets ayant de nouveaux comportements. La troisième phase introduit les structures de contrôles conditionnelles et itératives et les expressions booléennes. La quatrième phase se consacre aux méthodes et aux paramètres et la cinquième phase aux variables d'instances. Dans cette dernière phase les étudiants sont amenés à étendre des classes par l'ajout de nouveaux attributs membres. La structuration d'enseignements introductifs de la POO par l'usage de l'environnement ObjectKarel (variante de l'environnement Ka-

12. La fonction *main* est la fonction principale qui marque le début d'exécution d'un programme.

13. Hello world est un exemple de programme simple, dont le but est de faire une démonstration rapide d'un langage de programmation ou d'un compilateur. Souvent, il s'agit d'un programme qui affiche simplement à l'écran l'expression "*Hello world*".

rel) (cf. s.3.3.3.1) a été mise en expérience par Xinogalos, Satratzemi et Dagdilelis (2006). Les auteurs rapportent que les étudiants n'ont pas évoqué de difficultés liées aux concepts d'objets et de classes, que la moitié d'entre eux ont compris les concepts d'héritage et de polymorphisme et que la majorité d'entre eux ont développé des programmes correctes comprenant les structures de contrôles conditionnelles et itératives.

Joel Adams et Frens (2003) proposent une méthodologie d'introduction de la conception OO sur trois phases en utilisant le langage Java. Une première phase appelée "*Objets en premier*" consiste à apprendre aux étudiants à résoudre un problème en identifiant les objets qui doivent être déclarés dans un programme. Dans le cas où il n'est pas possible de représenter des objets en utilisant des types existants, il est recommandé de définir une nouvelle classe. Cette phase comprend également l'identification d'opérations devant s'appliquer aux objets. La seconde phase appelée "*Classes et méthodes*" se consacre à l'introduction de méthodes incluant leur déclaration et définition au sein d'une classe. Cette phase introduit, par la même occasion, les structures de contrôles conditionnelles et itératives afin d'élaborer les méthodes permettant de résoudre le problème de départ. La troisième phase appelée "*Héritage*" permet d'introduire les concepts d'héritage et de polymorphisme.

Cooper, Dann et Pausch (2003) ont adopté l'approche *Objets-D'abord* en utilisant l'environnement Alice (cf.s.3.3.3.2). Par cette démarche, ils proposent de réduire la complexité liée aux détails de la programmation que les débutants doivent surmonter, introduire les objets dans une approche conceptuelle et visualiser les objets dans un contexte significatif. L'utilisation de l'environnement Alice a été justifiée comme suit (Moskal, Lurie et Cooper, 2004) : *travailler avec un environnement 3D facile à utiliser est attrayant et engageant pour les générations actuelles qui possèdent une culture du numérique et du multimédia. La nature visuelle et le feedback immédiat des programmes aident les étudiants à facilement voir les résultats d'un ensemble d'instructions. Cela facilite également le débogage. L'éditeur Glisser-Déposer de l'environnement aide les étudiants à ne pas commettre des erreurs de syntaxes qui sont courantes chez les débutants. Les modèles 3D représentatifs de classes et d'objets dans Alice offrent une notion très concrète de ces concepts.*

2.6 Synthèse : POO, quel enjeu et quelle approche pour son enseignement et son apprentissage ?

2.6.1 *La place de l'OO dans l'industrie et l'académie*

Dès les années 1970, l'intérêt pour le paradigme OO a vite été développé, notamment avec les travaux d'Alan Kay, qui fut l'un des pionniers de la POO en définissant les premiers concepts de ce paradigme. Il fut l'un des premiers à ressortir un principe fondamental de l'OO, qui est la possibilité de représenter de façon modulaire un système complexe en décrivant ses propriétés tout en occultant ses détails. Il s'agit d'une façon de s'affranchir de la complexité inhérente à un système informatique. Ce principe, aujourd'hui appelé *modularité* lié à celui de *l'abstraction*, constitue un grand principe de qualité logicielle. Un principe qui, dès les années 1980, a suscité l'intérêt de plusieurs industriels, conduisant au développement de plusieurs langages de programmation en-

traînant plusieurs efforts de conceptualisation, dont les premiers datent notamment des années 1990.

Connaissant la place qu'occupe la POO dans l'industrie, la formation universitaire tente de s'adapter aux besoins et tendances de cette dernière. C'est pour cela qu'aujourd'hui la POO occupe une place importante dans la formation universitaire, et quelque soit l'approche d'enseignement adoptée, les notions de l'OO sont toujours introduites dans le curriculum.

2.6.2 *Nature des difficultés d'apprentissage de l'OO : divergence des avis*

Bien que l'OO occupe une place de plus en plus importante dans le curriculum universitaire, il apparait que, sur le terrain, l'enseignement et l'apprentissage de la POO ne se fassent pas sans difficultés.

Les travaux qui tentent de comprendre la nature et les causes de ces difficultés divergent dans leurs conclusions. Certains accusent l'abstraction inhérente aux concepts OO et aux relations existantes entres ces concepts. Ils justifient leur intérêt pour des outils de visualisation de programmes, dans l'intention d'amener les étudiants à mieux comprendre l'exécution en mémoire des programmes de haut niveau (Biju, 2013 ; Milne et Rowe, 2002 ; Nguyen et B. Wong, 2001). D'autres accusent plutôt des facteurs externes au paradigme de programmation lui-même, soutenant l'idée que l'OO est fondé sur des concepts naturels proches de notre perception du monde réel (Bashiru et Aderogba Joseph, 2015 ; Kölling, 2003 ; Kölling et Rosenberg, 2002). Ces facteurs sont notamment en lien avec les méthodes d'enseignement, les outils et les manuels employés qui échouent lorsqu'il s'agit de montrer l'intérêt de ce paradigme, en particulier, ses fondements naturels proches de l'expérience sensible ou du monde réel.

2.6.3 *Des activités de programmation concrètes et significatives : convergence des propositions*

Bien que les avis divergent quand il s'agit d'expliquer la nature et les causes des difficultés de l'apprentissage de la POO, les recherches visant à palier ces difficultés semblent converger (Becker, 2001 ; Cooper, Dann et Pausch, 2003 ; Kölling, 2010a). Il s'agit de proposer des solutions pouvant rendre le processus de programmation concret, en mettant l'accent sur les concepts avant l'implémentation dans un contexte réaliste et significatif. C'est dans ce contexte que l'approche *Objets-D'abord* s'est répandue dès les années 1990 en retenant l'attention de plusieurs chercheurs (Joel Adams et Frens, 2003 ; Bennedsen et Schulte, 2007 ; Cooper, Dann et Pausch, 2003 ; Woodworth et Dann, 1999). Il s'agit d'aborder de manière précoce les notions OO avant d'autres paradigmes, en particulier le procédural. En pratique, cette approche se déroule sur trois phases successives : *utilisation d'objets* (commencer par manipuler des objets préalablement implémentés), *création de classes* (déclaration et définition de nouvelles classes puis instanciation de ces classes) et *introduction de concepts* (initiation aux aspects conceptuels de l'OO). Les notions algorithmiques, de structures de données et de contrôle sont introduites au fur à mesure au travers des notions OO.

Bien que l'approche *Objets-D'abord* présente l'avantage de confronter les étudiants très tôt au paradigme OO et puisse remédier aux difficultés liées au changement de pa-

radigme (en particulier, de l'impératif vers l'OO), elle peut surcharger les étudiants avec un nombre important de concepts issus des deux paradigmes. Les spécificités des langages de POO tels que C++ et Java peuvent accentuer les difficultés d'apprentissage avec cette approche. C'est justement pour remédier à cela que les chercheurs qui soutiennent cette approche, utilisent des micromondes, environnements interactifs fondés notamment sur le principe de visualisation, tels que Karel, Alice et Greenfoot. Nous aborderons plus en détail ce type d'environnements dans le chapitre suivant. L'approche *Objets-D'abord* a été portée par l'usage de ce type d'environnements, dont le potentiel sur l'apprentissage a été observé. L'objectif étant d'aider les débutants à construire une compréhension fine des concepts abstraits de l'OO et de la faciliter, tout en fournissant des activités attrayantes.

2.6.4 *Une approche didactique par emboîtement hiérarchique des concepts fondamentaux de l'OO*

L'approche *Objets-D'abord* a permis d'introduire un concept qui n'est pas explicité, à notre connaissance, dans la littérature. Il s'agit de la manière avec laquelle cette approche est mise en application : 1) utilisation d'objets, 2) création de classes, 3) concepts (cf. s. 2.5.3). Avec cette approche, les concepts OO ne sont pas seulement introduits de façon progressive, mais d'une façon qui permet de se concentrer sur des notions particulières à une étape donnée, tout en occultant les détails liés à d'autres notions en lien avec les notions justement abordées. Ces notions occultées sont introduites directement à l'étape suivante. Cela permet de s'affranchir de la complexité inhérente aux concepts OO, qui sont très fortement liés les uns aux autres.

Nous proposons ici de montrer un nouveau concept, *une approche didactique par emboîtement hiérarchique des concepts fondamentaux de l'OO*. Nous voyons dans cette approche un emboîtement de trois catégories de concepts (Figure 6). La première catégorie contenant une seconde qui contient à son tour une troisième. Il s'agit à chaque étape d'interagir avec les propriétés d'une catégorie de concepts, les détails de chaque catégorie contenue étant occultée par une catégorie contenante.

Dans cette nouvelle approche didactique *l'emboîtement hiérarchique* permet l'introduction des fondamentaux de l'OO en trois étapes, comme suit :

1. Dans une première étape, il s'agit de créer et de manipuler des objets qui sont préalablement implémentés (*utilisation d'objets*), afin d'accéder aux notions liées au concept d'*objet*. Bien que le concept d'objet soit étroitement lié à celui de la classe, les principes de la classe sont volontairement occultés.
2. Dans une seconde étape (*création de classes*), il s'agit de déclarer et de définir de nouvelles classes en spécifiant leurs propriétés. C'est aussi une manière d'amener l'apprenant, après une première étape d'utilisation d'objets, à garder à l'esprit qu'une classe permet de définir les propriétés d'objets et que la modélisation a pour objectif de spécifier la solution à un problème, que l'on peut concrètement résoudre au moyen d'objets.
3. Dans une troisième étape, il s'agit de mettre l'accent sur les concepts de création de modèles OO complexes (*concepts*), telles que les relations d'interaction entre

classes. Une classe peut être liée à d'autres classes par des relations complexes. Ces dernières sont volontairement occultées lors de la seconde étape, puis explicitées lors de la troisième étape. Cela permet d'initier l'apprenant aux relations entre classes souvent complexes, après que le concept de classe ait été maîtrisé.

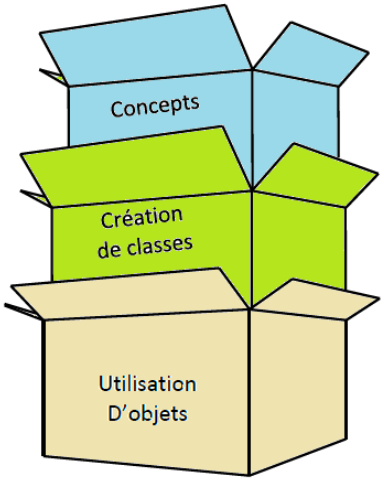


FIGURE 6 – Approche didactique par emboîtement hiérarchique des concepts fondamentaux de l'OO.

2.7 CONCLUSIONS ET IMPLICATIONS

Les recherches et travaux menés sur l'enseignement introductif de la programmation, et en particulier, l'introduction de la POO aux débutants révèlent que l'apprentissage est facilité par des outils visant à rendre le processus de programmation visuel et concret. C'est dans ce contexte que l'approche Objets-D'abord a été mise en pratique par plusieurs enseignants et a retenu l'attention de plusieurs chercheurs dans la communauté anglosaxonne. En pratique, cette approche a été portée par l'usage de micromondes tels que Karel, Alice et Greenfoot. L'objectif étant de mettre l'accent sur les concepts avant leur implémentation dans un langage de programmation. Cette approche vise à réduire les difficultés d'apprentissage de la programmation dans un contexte concret et ludique.

L'approche Objets-D'abord a permis d'introduire une nouvelle approche didactique pour l'apprentissage des concepts fondamentaux de l'OO. Elle permet d'introduire ces concepts de façon progressive, par emboîtement hiérarchique sur trois étapes importantes : 1) utilisation d'objets, 2) création de classes, 3) création de modèles OO. Cette approche emboîte les concepts de l'OO et les introduit de façon progressive, en mettant l'accent à une étape donnée sur certaines notions, tout en occultant les détails liés à d'autres notions en lien avec les notions justement abordées. Cette approche est en mesure de s'affranchir de la complexité inhérente aux concepts OO, qui sont très fortement inter-liés.

Cette approche a des implications importantes pour notre propre travail. Lorsqu'il s'agira de concevoir un nouvel artefact informatique pour l'introduction des fondamen-

taux de l'OO dédiés aux débutants, c'est cette approche didactique par emboîtement hiérarchique que nous retiendrons.

3

INITIATION À LA PROGRAMMATION PAR LE JEU

3.1 INTRODUCTION

Afin de répondre aux difficultés d'apprentissage de la programmation et de rendre la programmation de plus en plus accessible, plusieurs chercheurs ont contribué au développement d'une approche visant, d'une part, à rendre les concepts de programmation concrets et donc faciles à comprendre, et d'autre part, à rendre la programmation attrayante, et donc augmenter l'intérêt de l'apprenant pour cette discipline (Moskal, Lurie et Cooper, 2004 ; Papert, 1980). C'est dans cette perspective que plusieurs interfaces et environnements de programmation, reposant sur la programmation tangible et la programmation visuelle ont été conçus (Cooper, Dann et Pausch, 2000 ; Kölling, 2010b ; Perlman, 1976 ; Raffle, Parkes et Ishii, 2004). Il s'agit d'utiliser des objets physiques ou graphiques pour représenter les concepts de programmation.

La programmation tangible et la programmation visuelle sont nées, notamment avec les micromondes qui se caractérisent en plus de leur capacité à présenter les concepts abstraits de façon concrète, d'une dimension de *jeu-play* offrant des contextes significatifs et attrayants conduisant à la construction active des apprentissages.

Ce chapitre explore ces interfaces et environnements, qui présentent un réel potentiel dans l'apprentissage de la programmation (Papert, 1980) et plus particulièrement du paradigme OO (Xinogalos, Satratzemi et Dagdilelis, 2006). L'objectif étant d'identifier leurs fondements conceptuels offrant des activités significatives et attrayantes, conduisant à une compréhension facilitée de la programmation. Dans un premier temps, nous présenterons chacun des concepts de la programmation tangible (Section 3.2) et de la programmation visuelle (Section 3.3), au travers de leurs genèses et des interfaces et environnements de programmation qui les ont fait naître. Ensuite, nous présenterons le concept des micromondes (Section 3.4) qui incluent les environnements de programmation visuelle et les interfaces de programmation tangible. Nous décrirons leur genèse et les efforts de leur conceptualisation. Nous décrirons également leur rapport aux simulations et aux jeux, ainsi qu'à la programmation. Nous examinerons leurs principes conceptuels afin de comprendre et d'expliquer leur potentiel dans l'apprentissage de la programmation. Nous procèderons à une synthèse visant à tirer des conclusions sur l'apprentissage de la programmation par le jeu (Section 3.5). Enfin, nous conclurons sur les implications de ce travail sur le déroulement de la thèse (Section 3.6).

3.2 PROGRAMMATION TANGIBLE

3.2.1 *Définitions et genèse*

Selon McNerney (2004) l'expression "*Programmation tangible*" (*tangible programming*) a été introduite pour la première fois par Suzuki et Kato (1993) avec la conception

du système AlgoBlock, qui est un environnement de programmation collaboratif dédié aux enfants. Il consiste en un ensemble de blocs physiques qui s'assemblent les uns aux autres afin de constituer des programmes, où chaque bloc représente une commande exécutable. L'objectif étant de constituer des algorithmes physiques et de les exécuter virtuellement à l'écran. Il s'agissait de contrôler une représentation virtuelle de sous-marin (Figure 7).

FIGURE 7 – Le système AlgoBlock (SUZUKI et KATO, 1993).

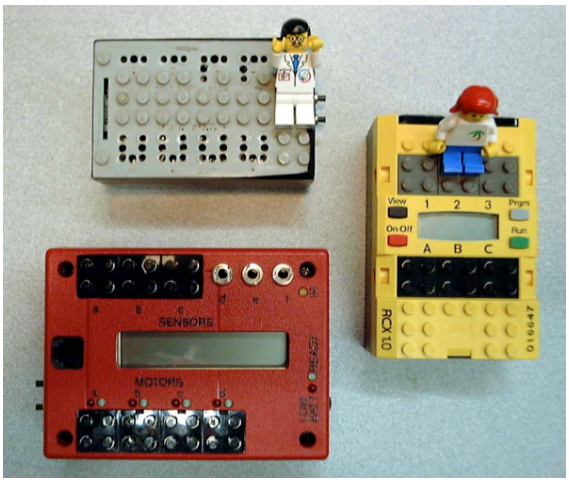
AlgoBlock a été alors introduit comme un environnement de programmation tangible. Il définit cette dernière comme *l'activité d'organiser des blocs physiques pour "construire" (par opposition à "écrire") des programmes informatiques* (SUZUKI et KATO, 1993).

Bien que ce terme fut utilisé pour la première fois par SUZUKI et KATO (1993), le concept de programmation tangible existait déjà bien avant. La programmation tangible supposait la manipulation d'objets physiques dans l'élaboration de programmes informatiques, plutôt que la saisie des commandes textuelles, mais dans certains cas ce sont les résultats du processus de programmation qui sont physiques et se produisent hors de l'écran d'ordinateur. C'est ainsi que depuis l'apparition du langage LOGO dès la fin des années 1960, qui permettait de contrôler les déplacements d'une tortue robotique et avec la publication d'un rapport de recherche au Massachusetts Institute of Technology (MIT) qui l'a fait connaître (PAPERT, 1980), plusieurs travaux ont été initiés au MIT afin de concevoir des interfaces de programmation physiques. Ces interfaces était alors basées sur LOGO, le système AlgoBlock conçu plus tard, est lui-même fondé sur LOGO (SUZUKI et KATO, 1993).

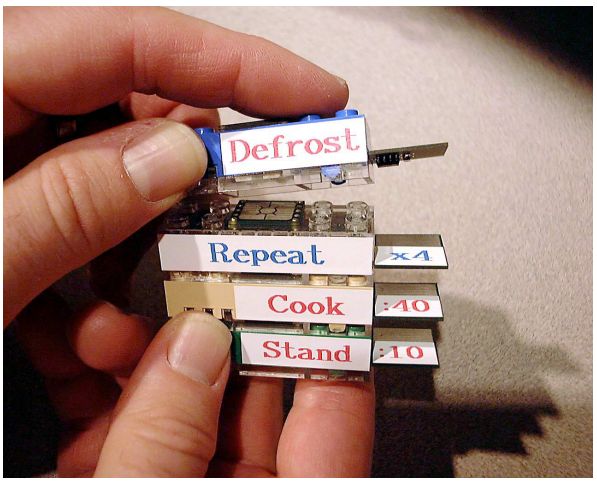
Cette période fut marquée par des recherches sur l'enseignement de la programmation aux enfants en âge préscolaire. Il paraissait alors évident qu'à cet âge, l'apprentissage de la programmation en saisissant du code dans une syntaxe de langage de programmation tel que LOGO était un réel obstacle. Il s'agissait de trouver d'autres interfaces de programmation qui permettaient aux plus petits d'apprendre à programmer tout en s'affranchissant d'un langage de programmation. C'est dans ce contexte que PERLMAN (1976) a proposé le système TORTIS qui permettait de programmer une représentation virtuelle de la tortue LOGO, en utilisant des commandes physiques comportant des symboles à la place d'instructions textuelles. Le système comportait une

boîte de commandes physiques et une machine à slot sur laquelle se plaçaient des cartes symboliques. Le tout était relié à un écran pour visualiser le comportement d'une tortue virtuelle.

Dans le milieu des années 1980, LOGO a été interfacé avec les produits du groupe LEGO [1] afin de permettre le contrôle de structures physiques constituées de pièces de LEGO avec des programmes LOGO implémentés sur ordinateur. Ceci a donné lieu au système *LEGO/LOGO* qui a été ensuite commercialisé par le groupe LEGO sous le nom de *LEGO tc logo* et qui a été utilisé en classe (Martin et al., 2000). Ce dernier a été étendu dès 1987, vers *les Briques Programmables* (*Programmable Bricks*) sur lesquelles il était possible de charger des programmes LOGO afin d'être exécutés sur les briques elles-mêmes (Figure 8a) (Martin et al., 2000). L'utilisateur constituait une structure LEGO sur laquelle la Brique sera portée, puis branchait la Brique Programmable à un ordinateur afin de programmer le comportement de la structure LEGO en utilisant le langage LOGO. Les Briques Programmables étaient équipées de capteurs sensoriels pour percevoir leur environnement et de moteurs de contrôle. Ce système a été ensuite commercialisé par le groupe LEGO sous le nom de *LEGO Mindstorms* qui continue à être produit aujourd'hui encore [2] (McNerney, 2004). Inspiré par le système AlgoBlock McNerney (1999) proposa les *Briques de Programmation Tangible* (*Tangible Programming Bricks*). Ce sont des briques électroniques qui s'emboîtent les unes sur les autres, représentant chacune une commande LOGO et possédant sur un côté un slot sur lequel s'insère une carte portant un paramètre de commande (Figure 8b). Ces briques sont munies d'une mémoire qui sauvegarde un programme LOGO. Chaque brique possède un microprocesseur qui exécute une commande LOGO et le programme donne à chaque brique un comportement unique. Ce système a été testé pour le contrôle de jouets physiques (McNerney, 1999).



(a) Les Briques Programmables (dès 1987) (Martin et al., 2000).



(b) Les Briques de Programmation Tangible (McNerney, 1999).

Figure 8 – Interfaces tangibles programmables avec LOGO.

1. LEGO est une marque déposée du groupe LEGO (http://www.lego.com/)
2. http://www.lego.com/fr-fr/mindstorms

En vue de rendre la programmation de plus en plus accessible aux plus jeunes, d'autres interface ont été conçues jusque dans les années 2000 notamment avec la conception du système Topobo (Raffle, Parkes et Ishii, 2004). Il s'agit d'un jeu de construction et d'animation entièrement tangible. Il a été conçu dans le but de permettre à l'utilisateur d'effectuer des transitions entre leurs connaissances manuelles et kinesthésiques et des connaissances computationnelles qui peuvent être reformulées et testées (Raffle, Parkes et Ishii, 2004). Ce système introduit un nouveau modèle computationnel, dans lequel le processus de programmation et le résultat de son exécution sont entièrement tangibles. Cela vient donc étendre le concept de la programmation tangible qui n'est pas seulement inhérent aux interfaces permettant l'élaboration (sans passer par un langage de programmation) ou l'exécution (hors de l'ordinateur) de programmes de façon physique, mais aussi aux interfaces permettant à la fois d'établir et de visualiser physiquement le résultat d'exécution de programmes. Dans la section suivante, nous décrirons avec plus de détails l'environnement LOGO (tangible dans le résultat du processus de programmation), TORTIS (tangible dans le processus de programmation) et Topobo (tangible à la fois dans le processus de programmation et dans le résultat d'exécution de programmes).

3.2.2 *Descriptions de quelques interfaces de programmation tangible*

3.2.2.1 *L'environnement LOGO*

LOGO est à la fois un langage de programmation et un environnement d'apprentissage de la géométrie dédié aux enfants (Papert, 1980). Le langage LOGO est constitué de commandes permettant de contrôler les déplacements d'un robot tortue, en dessinant des formes géométriques en résultat aux mouvements de la tortue. Le robot est muni d'un stylo et se déplace sur le sol en dessinant des formes géométriques (Figure 9). LOGO se base sur un principe appelé *géométrie de la tortue*, que Papert (1980) définit

Figure 9 – La tortue robotique de LOGO.

comme une *approche computationnelle de la géométrie*, ayant pour concept fondamental la *tortue* : *"le concept de la tortue est en lien avec ce qu'on connaît. Contrairement à un concept abstrait, elle possède des propriétés, une apparence dynamique et la capacité de comprendre un*

langage de commandes" (Papert, 1980). Étant à une position donnée, elle est capable de se déplacer dans n'importe quelle direction en réponse à une commande. De ce fait, elle sert de représentation de concepts mathématiques formels.

Le langage LOGO, appelé *le langage de la tortue* (Papert, 1980), est constitué de commandes telles que `FORWARD et BACK` qui font déplacer la tortue en ligne droite en avant ou en arrière. Ces commandes suivies d'un chiffre font déplacer la tortue d'un nombre de pas exprimé par ce chiffre. Les commandes `RIGHT` et `LEFT` permettent de faire tourner la tortue à droite ou à gauche de sa direction courante. Suivies de chiffres, ces commandes permettent de tourner la tortue à droite ou à gauche d'un angle exprimé par ces chiffres. Ainsi, un carré peut être produit par les commandes suivantes :

```
TO SQUARE
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
FORWARD 100
RIGHT 90
END
```

Ceci permet alors de constituer des programmes et des procédures. L'apprentissage de la géométrie passe alors par l'apprentissage de la programmation. Papert (1980) indique que cela permet d'apprendre l'organisation hiérarchique de connaissances, la planification et le débogage. À titre d'exemple, il est possible de constituer la procédure `TRIANGLE` comme suit :

```
TO TRIANGLE: SIDE
REPEAT 3
 FORWARD: SIDE
 RIGHT 120
END
```

Ceci donne la possibilité de dessiner des triangles de différentes tailles en formulant les commandes : `TRIANGLE 50` ou `TRIANGLE 100`. Cela permet d'organiser le programme en procédures qui constituent des parties de programmes réutilisables. Le débogage résulte du fait qu'il est possible de visualiser le résultat des commandes, puis de réajuster le programme dans le cas où le résultat ne correspond pas à ce qui a été prévu. À titre d'exemple, soit le programme constitué des commandes suivantes :

```
TO HOUSE
SQUARE
TRIANGLE
END
```

Alors que le résultat souhaité serait de dessiner le triangle au-dessus du carré, ce programme affichera le triangle à l'intérieur du carré. Afin d'arriver au résultat voulu, l'apprenant va d'abord insérer la commande `RIGHT 30` entre `SQUARE` et `TRIANGLE`. Il verra que le résultat s'est amélioré, mais ne correspond pas tout à fait à ce qu'il voulait. Avec un second essai, en insérant `RIGHT 90` à la place de `RIGHT 30`, il arrivera finalement à corriger son erreur.

3.2.2.2 *Le système TORTIS*

TORTIS est un système de programmation tangible destiné aux enfants en âge préscolaire (Perlman, 1976). Il a permis à des enfants âgés de 3 à 5 ans de programmer le comportement de la tortue LOGO, en utilisant un sous langage de LOGO basé sur des symboles à la place de commandes textuelles. Le système TORTIS est constitué de deux parties : *la boîte à boutons* (*Button box*) et la *machine à slots* (*Slot Machine*). *La boîte à boutons* était constituée de quatre boîtes qui se branchent les unes aux autres, où chaque bouton représente une commande. La *machine à slots* était constituée de plusieurs lignes rectangulaires en plexiglas représentant chacune une procédure.

TORTIS a été conçu de sorte à simplifier la programmation aux plus jeunes (Perlman, 1976). La boîte à boutons est modulaire et permet à l'utilisateur d'être introduit progressivement aux différentes commandes (Figure 10).

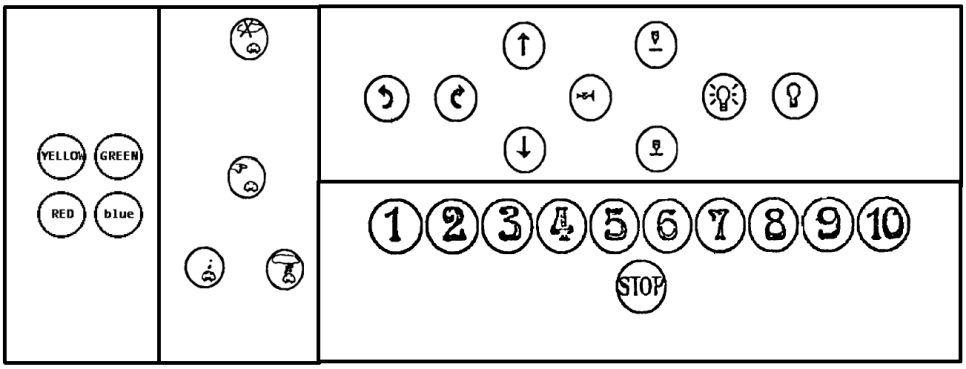


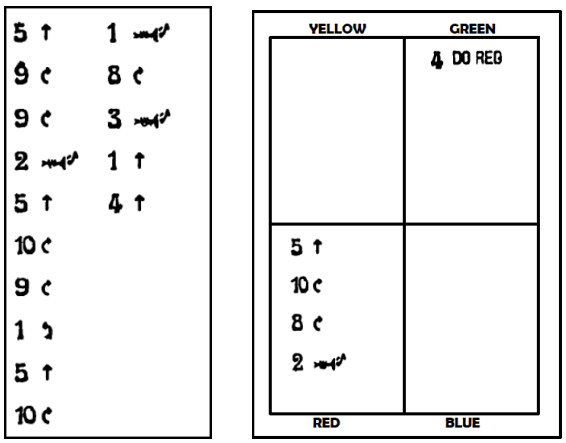
Figure 10 – La boîte à boutons du système TORTIS (Perlman, 1976).

La première boîte à boutons est appelée *boîte d'actions* (*action box*). Elle est constituée de commandes permettant de contrôler le mouvement de la tortue virtuelle à l'écran dans un langage impératif : `forward` (en avant), `backward` (en arrière), `rotate clockwise (right)` (tournes à droite dans le sens des aiguilles d'une montre), `rotate clockwise (left)` (tournes à gauche dans le sens des aiguilles d'une montre), `toot your horn` (déclenches un avertisseur sonore), `put the pen down onto the paper` (places le stylo sur le papier), `lift the pen off the paper` (retires le stylo du papier), `turn your light on` (allumes la lumière) et `turn your light off` (éteints la lumière). À chaque fois qu'un bouton est appuyé, le résultat est immédiatement visible sur la tortue.

La seconde boîte est la *boîte de nombres* (number box) qui contient des nombres de 1 à 10 ainsi que la commande "`STOP`" qui fait interrompre une action en cours d'exécution. Appuyer sur un chiffre avant une action, répète l'action un nombre de fois déterminé par ce chiffre (Figure 11a).

La troisième boîte est la *boîte à mémoire* qui possède quatre commandes impératives : `start remembering` (commences à te rappeler), `stop remembering` (cesses de te rappeler), `do it` (fais-le) et `forget it` (oublies-le). La boîte à mémoire permet la sauvegarde d'un ensemble de commandes et leur réexécution. Elle est utilisée avec un écran d'ordinateur, permettant d'afficher des commandes constituées et mémorisées. À titre d'exemple, quand le bouton "`start remembering`" est appuyé, l'écran d'affichage s'allume et se met en attente d'actions d'utilisateur. Quand un bouton action est appuyé, une image correspondante s'ajoute à une liste de commandes qui s'affiche à l'écran.

La dernière boîte, *la boîte à quatre procédures* (four procedure box), est constituée de quatre boutons couleurs (`Yellow, Green, Red, Blue`), permettant de constituer des procédures, dont le contenu s'affiche à l'écran. Quand cette boîte est actionnée, l'écran affiche quatre cadrans colorés suivant les quatre couleurs. Quand un bouton mémoire est appuyé, il est alors indispensable d'actionner un bouton couleur afin de choisir le cadran de l'écran sur lequel les commandes à mémoriser, à exécuter ou à effacer vont s'afficher. Le bouton couleur peut constituer la commande "*fais couleur*" à l'intérieur de chacune des quatre procédures. Cela permet de programmer des sous procédures (Figure 11).



(a) Procédure unique.

(b) Utilisation de la boîte à quatre procédures.

FIGURE 11 – Exemples de programmes TORTIS pour le contrôle de la tortue robotique de LOGO (PERLMAN, 1976).

La machine à slots est constituée de lignes rectangulaires dans lesquelles, il est possible d'insérer des cartes électroniques imagées représentant les différentes commandes du système. La constitution de procédures revient alors à insérer différentes cartes dans les connecteurs présents sur les lignes de la machine (Figure 12). L'exécution des procédures est visible sur le comportement de la tortue LOGO à l'écran d'un ordinateur.

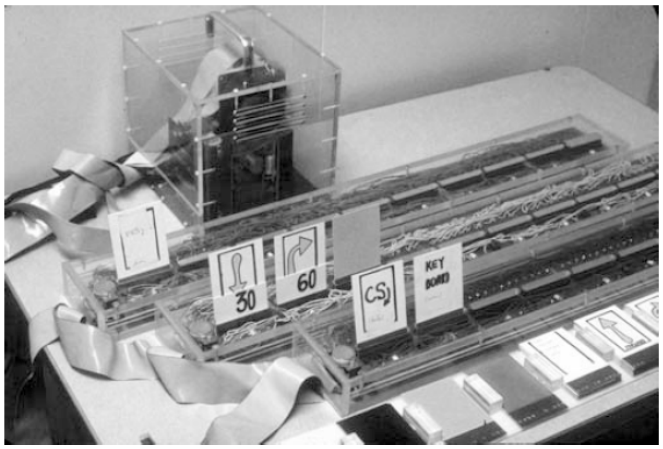


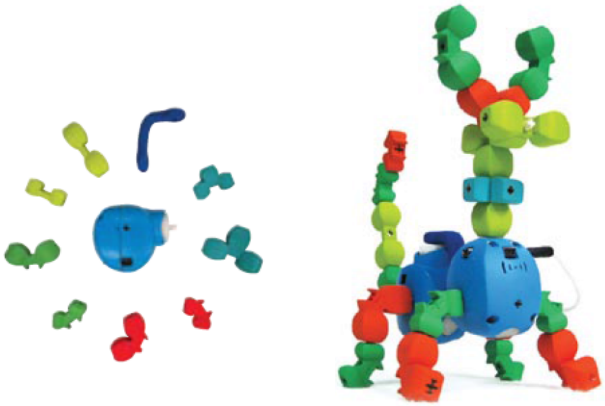
FIGURE 12 – La machine à slots du système TORTIS (1976) (MCNERNEY, 2004).

3.2.2.3 *Le système Topobo*

Topobo est un jeu de construction et d'animation tangible dédiés aux enfants âgés de 5 à 13 ans (Raffle, Parkes et Ishii, 2004). Il permet d'assembler des composants physiques afin de constituer des structures robotiques, puis de les animer en appliquant des mouvements physiques sur ces composants (Figure 13).

Topobo introduit un nouveau style de programmation. Il se programme grâce à la mémoire cinétique de ses composants, sans recourir à des structures à base de blocs de commandes langagières. La programmation Topobo est fondée sur la kinesthésie et la capacité de l'enfant à réaliser une action corporelle. Ce système est décrit comme un système computationnel qui emploie la Kinesthésie comme moyen de représentation de concepts abstraits (Raffle, 2008). Il part de l'idée que les interfaces tangibles sont intuitives et très simples, tandis que les langages de programmation sont sophistiqués et difficiles à apprendre par les plus jeunes. Il a été alors conçu dans l'objectif de permettre à l'utilisateur de passer de concepts intuitifs et faciles à des concepts complexes et abstraits : " *un tel système doit présenter l'avantage d'introduire à la fois la conception et la programmation. Il doit également permettre le renforcement de représentations abstraites et de notions computationnelles*" (Raffle, 2008).

Le système se compose de dix différentes primitives qui s'assemblent les unes aux autres (Figure 13a), dont neuf *passives*, qui constituent des connections statiques et une *active*, munie d'un moteur à mémoire cinétique. Les composants motorisés sont les seuls à avoir la capacité de se mettre en mouvement. Le système est capable de sauvegarder et de restituer toute manipulation dynamique appliquée sur une structure constituée.



(a) Les dix primitives de Topobo.

(b) Constitution de structures physiques animées.

Figure 13 – Le système Topobo (Raffle, 2008).

Les composants passifs sont conçus sous différentes formes afin de permettre la constitution de structures physiques variées. Les composants actifs sont munis de Diodes ÉlectroLuminescente (DEL), indiquant si le système est en mode *sauvegarde* (lumière rouge) ou en mode *restitution* (lumière verte). Afin d'enregistrer un mouvement, il suffit d'appuyer sur un bouton dédié, disposé sur un composant actif, puis d'appliquer un mouvement sur le composant afin de programmer une séquence de mouvement

(Figure 14). Appuyer de nouveau sur ce bouton met fin à l'enregistrement du mouvement et fait basculer le composant en mode restitution. Celui-ci rediffuse le mouvement sauvegardé de façon répétitive jusqu'à ce que le bouton soit appuyé une troisième fois.

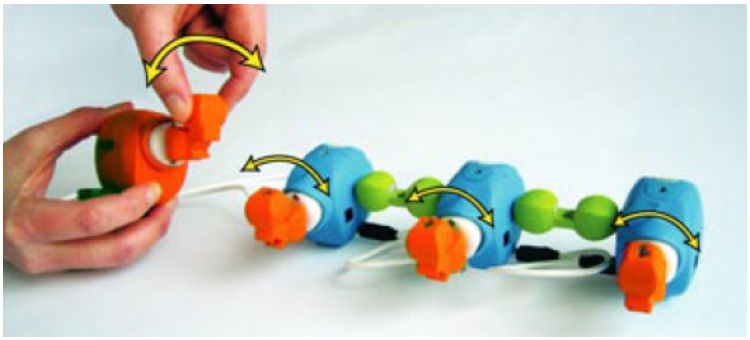


FIGURE 14 – Principe de programmation tangible dans Topobo (RAFFLE, 2008).

Le couplage de constructions physiques et de la programmation gestuelle fait de Topobo une interface de programmation entièrement tangible. RAFFLE (2008) explique que le fait d'assembler différents composants actifs dans une création unique, est équivalent à créer des programmes parallèles qui sont synchronisés dans le temps. Le couplage du mouvement avec un système 3D permet aux plus jeunes d'explorer les interactions entre des programmes parallèles simples et des mouvements physiques complexes. Néanmoins, il reconnait dans Topobo une forme de programmation limitée. Bien que Topobo soit capable d'introduire des notions computationnelles abstraites, le système n'a pas la flexibilité des langages de programmation (RAFFLE, 2008). Alors que le système permet d'introduire les concepts de *boucles et de fonctions*, la structure de contrôle de son interface se retrouve limitée du fait de sa nature entièrement physique. Le système offre la possibilité de créer des mouvements complexes et coordonnés, sans pouvoir les contrôler de manière sophistiquée, comme pourrait le faire un langage de programmation. RAFFLE (2008) précise également que cette limite pourrait être surmontée par l'interfaçage de Topobo avec des langages de programmation visuelle.

Dans ce qui suit nous décrirons plus en détail la programmation visuelle, à travers quelques environnements visuels dédiés à la programmation impérative puis à la POO.

3.3 PROGRAMMATION VISUELLE

3.3.1 *Définitions et genèse*

Une taxonomie de la programmation visuelle et de la visualisation de programme et la genèse des langages visuels sont données par MYERS (1990). Il définit la programmation visuelle comme se référant à *tout système qui permet à l'utilisateur de spécifier un programme en deux (ou plus) dimensions, par opposition aux langages de programmation textuels qui permettent d'élaborer des programmes compilés ou interprétés en flots d'une seule dimension*. La programmation visuelle selon cet auteur, n'inclue pas les systèmes qui utilisent des lignes de commandes textuelles pour définir des graphiques, ou des systèmes qui incluent des bibliothèques de dessins. Cette définition s'accorde avec celle donnée par BEGEL (1996), qui désigne la programmation visuelle ou graphique comme

l'utilisation d'expressions graphiques et symboliques, ainsi que des animations dans le processus de programmation.

Contrairement à la programmation visuelle, la visualisation de programme sous-entend la rédaction du programme dans un langage textuel et l'utilisation de graphiques pour illustrer certains aspects du programme ou son exécution (Myers, 1990). Cela peut illustrer le code, les données ou les algorithmes de façon à montrer une animation du programme en exécution (visualisation dynamique), ou des captures imagées du programme à certains moments de son exécution (visualisation statique). Myers (1990) explique que la programmation visuelle doit se faire de manière graphique et représente d'une certaine manière une forme de visualisation de programme. Cependant, il recommande d'utiliser le terme programmation visuelle, pour désigner des systèmes permettant de créer des programmes de manière graphique et de réserver le terme de visualisation de programme, pour désigner des systèmes utilisant des graphiques pour illustrer des programmes après qu'ils aient été créés.

À la notion de programmation visuelle et de visualisation de programme est rattachée la notion de langages visuels, qui se réfère selon Myers (1990) à tout système qui utilise des graphiques, incluant à la fois la programmation visuelle et la visualisation de programme.

Selon Myers (1990), la programmation visuelle remonte à la fin des années 1960 avec l'apparition du langage *Grail* (Ellis, Heafner et Sibley, 1969), qui permettait de générer des programmes directement à partir d'organigrammes. Depuis, une multitude d'autres langages ont été conçus, tel que *Pygmalion* (Smith, 1975) qui était un langage iconique avec lequel l'utilisateur peut créer, modifier et assembler des objets picturaux pour constituer des programmes. On peut également citer *FPL* (First Programming Language) (Cunniff, Taylor et J. B. Black, 1986) qui était présenté comme particulièrement adapté pour aider les novices à apprendre la programmation.

C'est justement ce potentiel dont dispose ce type de langages qui a motivé les recherches dans ce domaine. Myers (1990) affirme que l'une des raisons pour lesquelles l'intérêt pour la programmation visuelle et la visualisation de programme s'est développé, réside dans le système visuel humain qui est naturellement optimisé quand il s'agit de traiter des informations multidimensionnelles. Les programmes informatiques écrits en langages textuels sont unidimensionnels et n'utilisent pas pleinement la capacité du cerveau. Très tôt, plusieurs auteurs démontrent que les langages visuels aident et facilitent la compréhension de la programmation (M. H. Brown et Sedgewick, 1984 ; Myers, Chandhok et Sareen, 1988). Clarisse et Chang (1986) affirment que la programmation visuelle utilise l'information dans un format qui se rapproche des représentations mentales que l'utilisateur se fait des problèmes. Cela permet de traiter les données dans un format proche de la manière dont les objets sont manipulés dans le monde réel. Une autre raison qui explique la motivation pour l'usage des langages visuels, réside dans la manipulation directe de l'interface utilisateur des éditeurs de programmation visuelle (Shneiderman, 1981). Ces interfaces permettent de sélectionner des items à l'écran directement via la souris. Cela donne à l'utilisateur l'impression de construire un programme de manière plus directe, plutôt que de le concevoir de manière abstraite (Shneiderman, 1981).

Les travaux qui se sont intéressés aux langages visuels en vue de rendre la programmation attrayante et facile à appréhender par les débutants, ont conduit au développement d'environnements visuels pour l'apprentissage de la programmation dans un contexte de jeu et d'animation (Begel, 1996 ; Bergin, J. Roberts et al., 1997 ; M. Conway et al., 2000 ; Kölling, 2010a ; Muratet, 2010 ; Resnick et al., 2009). C'est le cas de l'environnement *LogoBlocks* (Begel, 1996), qui fut l'un des premiers environnements de programmation visuelle dédié au paradigme impératif, dont le résultat d'exécution des programmes était visualisé sur des structures tangibles hors de l'ordinateur. C'est aussi le cas de *Scratch* qui est très largement utilisé aujourd'hui pour initier les plus jeunes à la programmation impérative (Resnick et al., 2009). Il s'agit à la fois d'un environnement de programmation visuelle et de visualisation de programmes. *Prog&Play* est un autre exemple de systèmes développés dans ce contexte. Il permet de visualiser le résultat d'exécution de programmes élaborés dans des EDIs, dans des missions de jeux de Stratégie Temps Réel (STR) (Muratet, 2010).

Dans le contexte de la POO, les environnements les plus populaires sont *Karel* (Bergin, J. Roberts et al., 1997), *Alice* (M. Conway et al., 2000) et *Greenfoot* (Kölling, 2010a). Dans les sous-sections suivantes, nous décrirons plus en détails ces deux catégories d'environnements visuels. Les premiers étant dédiés au paradigme impératif et les second dédiés au paradigme OO.

3.3.2 *Descriptions de quelques environnements visuels dédiés à la programmation impérative*

3.3.2.1 *LogoBlocks*

LogoBlocks est un environnement de programmation visuelle (Begel, 1996), qui a été conçu au MIT pour être utilisé avec les Briques Programmables (cf. s. 3.2.1). Son objectif était d'offrir aux plus jeunes un environnement de programmation plus aisé et plus amusant (Martin et al., 2000). Il se base sur le langage BrickLogo, qui est lui-même fondé sur le langage LOGO auquel sont ajoutées des primitives de contrôle de capteurs et de moteurs (Begel, 1996). À titre d'exemple, le programme BrickLogo suivant met en marche le moteur `a`, inverse la direction du moteur `b`, puis met en marche le moteur courant pendant une période de temps, enfin, répète 4 fois la mise en marche du moteur courant pendant 0.5 secondes puis l'arrête pendant 0.5 secondes :

```
1 a,on
  b,rd
  onfor <time period>
  repeat 4 [onfor 5 wait 5]
```

Le langage LogoBlocks est une variante graphique du BrickLogo. L'utilisateur constitue des programmes dans une interface graphique en glissant-déposant des blocs visuels colorés symbolisant des commandes BrickLogo. Le programme graphique est ensuite analysé, compilé, puis traduit en code postfix pour être chargé et exécuté sur les Briques Programmables (Begel, 1996).

Le langage LogoBlocks est constitué de différents types de blocs : *des blocs actions, des blocs capteurs et des blocs variables*. Les blocs actions (e. g. `wait et repeat`) permettent de contrôler les moteurs. Les blocs capteurs permettent de réaliser des contrôles condi-

tionnels sur l'état des capteurs (e. g. `if sensora < 45 [onfor 5]`). Les blocs variables permettent de donner des paramètres chiffrés aux fonctions qui les utilisent (e. g. `onfor`, `wait et repeat`).

LogoBlocks permettait la constitution de procédures imbriquées. La Figure 15 montre un exemple de programme qui contrôle les déplacements d'un jouet (voiture LEGO) muni de deux capteurs frontaux. Le programme implémente une procédure d'évitement d'obstacles : le LEGO doit se déplacer jusqu'à la détection d'un obstacle, faire demi-tour pendant 2.5 secondes, tourner pendant 2.5 seconds, puis reprendre son déplacent jusqu'à ce qu'un obstacle soit détecté de nouveau.

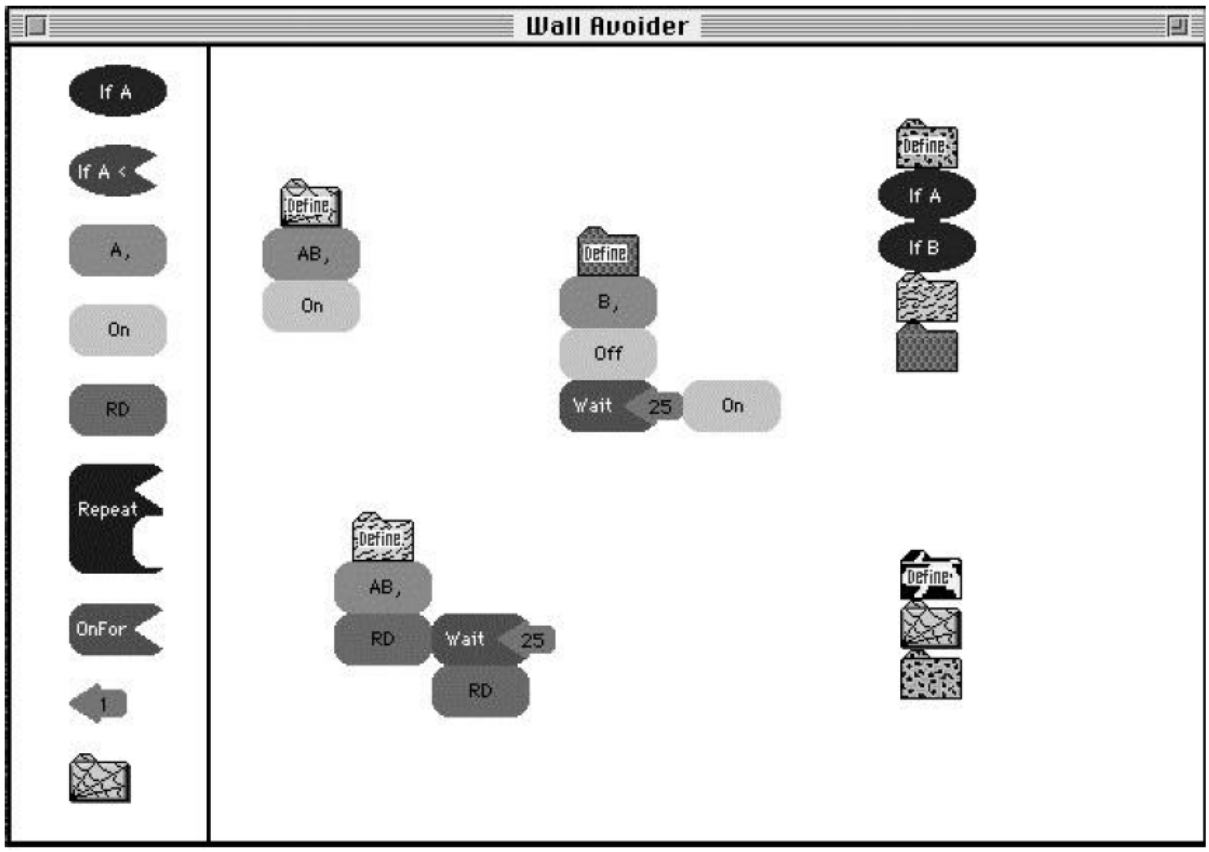


FIGURE 15 – L'interface LogoBlocks montrant un exemple de programme visuel (BEGEL, 1996).

3.3.2.2 *Scratch*

Scratch[3] est un environnement de programmation entièrement visuel, dont l'objectif est de rendre la programmation procédurale facile à comprendre par les plus jeunes (RESNICK et al., 2009). Il a été conçu sur la base des principes apportés par LOGO et d'autres environnements de programmation visuelle conçus avant lui, tel que Alice (cf. s. 3.3.3.2), mais se veut plus créatif, plus significatif et plus social que les autres environnements (RESNICK et al., 2009).

Développé au MIT, la grammaire du langage Scratch se base sur un ensemble de blocs graphiques qui s'enclenchent les uns dans les autres pour créer des programmes

3. https://scratch.mit.edu

(Figure 16). Cela permet à l'utilisateur de s'affranchir de toute difficulté liée à la syntaxe d'un langage de programmation textuel. Les blocs graphiques sont conçus de façon à permettre leur enclenchement en respectant la syntaxe grammaticale du langage. Les structures de contrôle comme "`Répéter`", possèdent une forme qui invitent à placer d'autres blocs à l'intérieur. La forme des blocs qui retournent des valeurs est définie selon le type de la valeur à retourner (formes ovales pour les nombres et hexagonales pour les booléens). Les structures conditionnelles comme "`si`" ou "`si, sinon`" incluent des formes hexagonales indiquant qu'un booléen est requis.

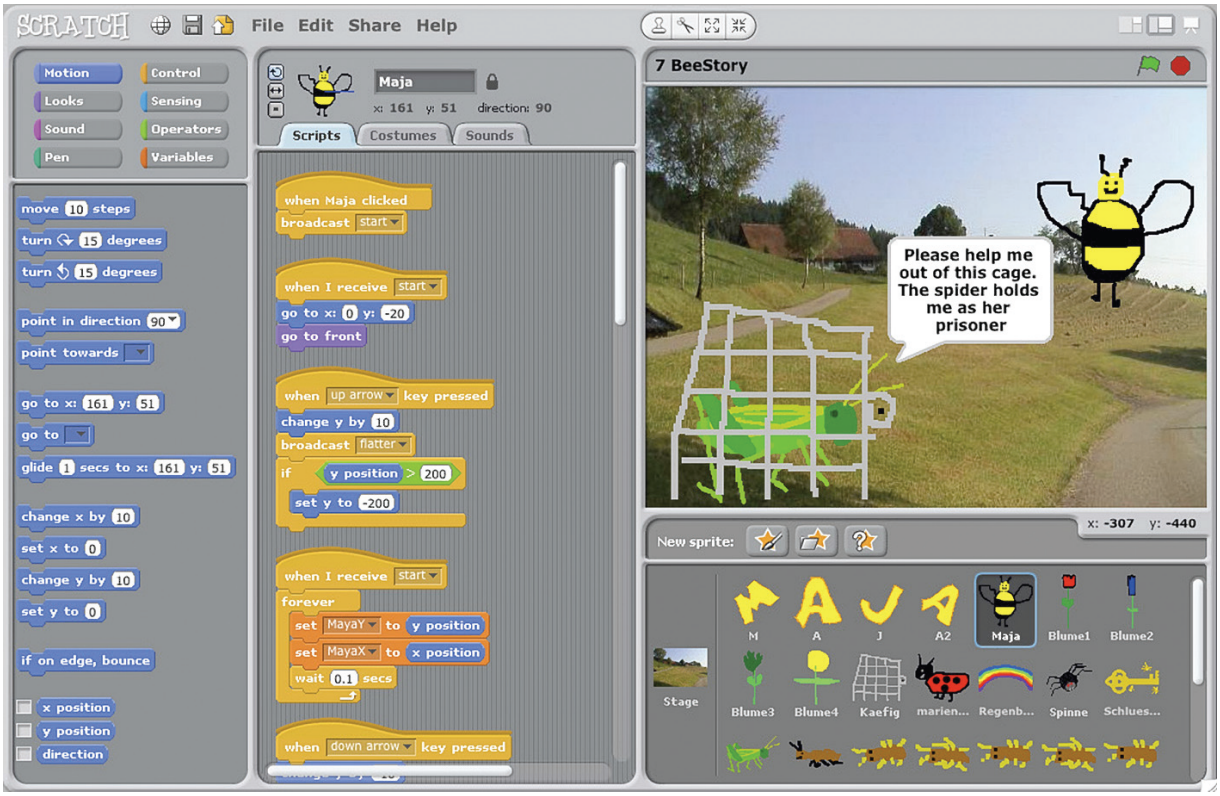


FIGURE 16 – L'environnement Scratch montrant un exemple de script.

Scratch se fonde sur "*le principe de bricolage*" (en anglais, *tinkering*). RESNICK et al. (2009) expliquent que le terme anglais *Scratch* a été utilisé en référence à l'idée de *bricoler* et à une technique musicale apparue avec le *rap* (*scratching*). Cette technique consiste à placer un disque vinyle sur une platine, puis faire aller le bras de lecture en avant et en arrière le long du sillon. Le but étant d'obtenir un son destiné à habiller un morceau de musique *rap* et de réaliser des mixtures musicales de façon créative. C'est par analogie à cette technique musicale que les concepteurs de Scratch présentent le processus de programmation comme une activité *qui mixe des graphiques, des animations, des images et des sons* (RESNICK et al., 2009).

Le principe *de bricolage* sur lequel Scratch est fondé, découle probablement de *la métaphore de bricolage*, un terme introduit par *Claude Lévi-Strauss*, puis repris par PAPERT (1980) pour désigner le processus d'apprentissage avec LOGO qui est *semblable à celui de bricolage* (*tinkering-like process*) :" *ce processus rappelle celui de bricolage ; l'apprentissage consiste à cumuler un ensemble de matériaux et d'outils qu'on peut s'approprier et manipuler*" (PAPERT, 1980). *Bricoleur* est le terme désignant celui qui s'engage dans une activité de

bricolage. Des principes qui ont été également repris dans la thèse de Lawler (1979) dans un contexte d'apprentissage de la programmation.

Un autre principe de Scratch est la mise en place d'une expérience significative, fondée sur la diversité des projets (jeux, narrations, animations et simulations) et la possibilité de les personnaliser (importer des images, des sons et créer de nouveaux graphiques). Enfin, Scratch est fondé sur l'existence d'une communauté où les usagers peuvent partager leurs réalisations et collaborer à la création de nouveaux projets. Le principe du partage est implémenté au sein même de son interface comportant un menu "*Partage*" (*Share*).

3.3.2.3 *Prog&Play*

Prog&Play a été conçu dans le but d'interfacer des moteurs de jeux de STR avec des EDIs (Muratet, 2010). L'objectif du système Prog&Play est de permettre à l'apprenant de réaliser des programmes capables d'interagir avec un jeu de STR. De cette manière, l'apprenant pourra suivre le déroulement de son programme à travers le comportement des unités du jeu vidéo. Prog&Play utilise des moteurs de jeu qui proposent des langages de script et des interfaces associées permettant la modification et la personnalisation de jeux. Néanmoins, ces langages sont inadaptés à l'introduction de la programmation aux débutants, d'où l'intérêt de Prog&Play pour des langages plus appropriés tels que C, C++, Java, etc. (Muratet, 2010).

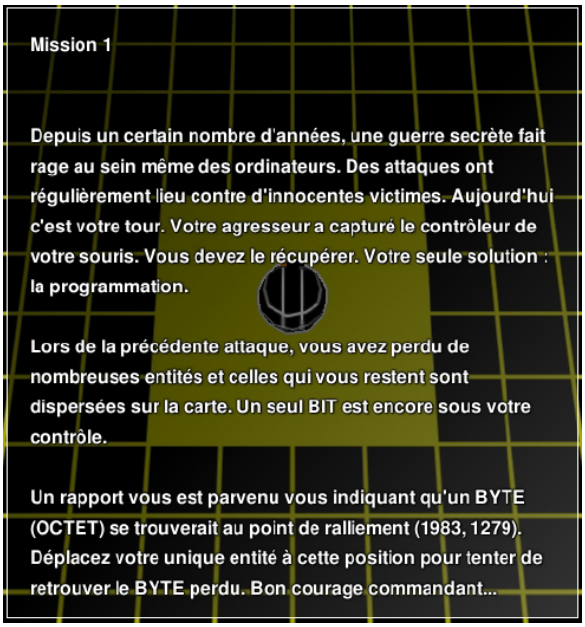
Prog&Play est disponible sous la forme d'une Application Programming Interface (API)[4] utilisable pour la conception de programmes compatibles avec certains jeux de STR qui sont intégrés dans le système. Il a été intégré à plusieurs moteurs de jeux de STR (e. g. *ORTS, Spring et WarZone 2100*), fournissant des outils de pratique de la programmation dans différents langages. Le jeu sérieux *Kernel Panic Campaign* a été alors réalisé en intégrant Prog&Play aux moteur de jeu *Spring*, en se basant sur un jeu déjà existant, *Kernel Panic*, dont l'univers du jeu est un environnent 3D constitué d'unités d'ordinateur (e. g. bit, octet, etc.).

L'API Prog&Play est développée en langage C et offre deux interfaces : *Interface Client*, qui est utilisée par le joueur pour développer un programme et *Interface Fournisseur*, qui est intégrée au moteur du jeu. Sur demande du programme du joueur, les données pertinentes du jeu sont copiées dans une mémoire partagée. La partie *Fournisseur* de cet API crée et initialise la mémoire partagée de manière à ce qu'elle soit accessible par la partie *Client*. Du côté utilisateur, l'appel aux fonctions de l'API (e. g. `PP_GetUnitAt()`, `PP_Unit_GetPosition()`, `PP_Unit_ActionOnPosition()`, etc.) peut être réalisé après s'être connecté à la mémoire partagée et avoir demandé un premier rafraîchissement.

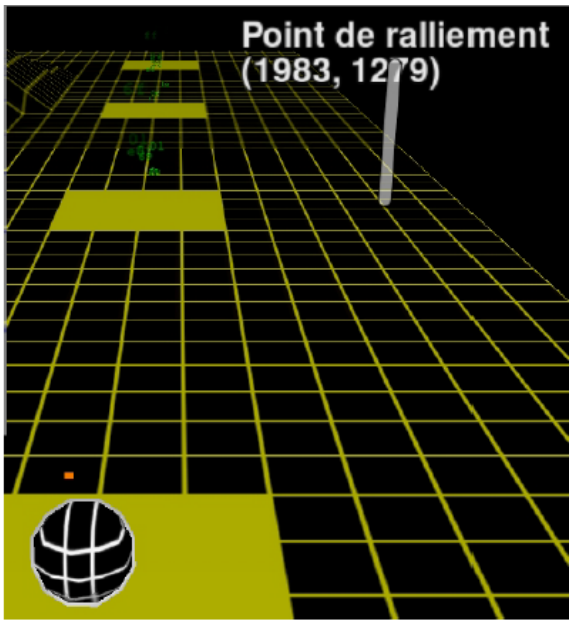
Le jeu *Kernel Panic Campaign* est structuré autour d'un scénario divisé en sept missions, qui posent chacune un problème que l'apprenant doit résoudre par un programme informatique. Ce scénario n'autorise pas l'utilisation de la souris. Le contrôle du jeu se fait uniquement par la programmation. Ce scénario est conçu de sorte à permettre d'introduire progressivement un concept algorithmique dans chaque mission : la séquence est introduite lors de la première mission, la structure de contrôle conditionnelle lors de la troisième, la structure de contrôle itérative lors de la quatrième et les

4. Prog&Play a été développé à l'université de Toulouse (France) : www.irit.fr/ProgAndPlay

structures de contrôle imbriquées lors de la sixième (Muratet, 2010). À titre d'exemple, lors de la première mission, le joueur doit déplacer, à une position indiquée, une seule unité de jeu (Figure 17).



(a) Explication de la mission.



(b) Problème à résoudre par la programmation.

Figure 17 – Mission 1 du jeu *Kernel Panic Campaign*.

La réalisation de la première mission de ce scénario se traduit par la réalisation d'un programme qui utilise des variables, des types, des affectations, des fonctions et le passage de paramètres. Le programme doit, dans un premier temps, inclure l'interface *Client* (`PP_Client.h`) ainsi qu'une liste de constantes de *kernel Panic* en C (`constantList_KP3.8.h`), puis ouvrir l'API Prog&Play (`PP_Open()`) et demander une mise à jours de données en mémoire partagée (`PP_Refresh()`). Le programme doit se terminer par la fermeture de l'API (`PP_Close()`). L'implémentation en langage C de ce programme est donnée comme suit (Muratet, 2010).

```
#include "PP_Client.h"
#include "constantList_KP3.8.h"

int main (void){
PP_Pos p;
PP_Unit u;
p.x = 1983.0;
p.y = 1279.0;
PP_Open();
PP_Refresh();
u = PP_GetUnitAt(MY_COALITION,0);
PP_Unit_ActionOnPosition(u,MOVE,p);
PP_Close();
return 0;
}
```

3.3.3 *Descriptions de quelques environnements visuels dédiés à la POO*

3.3.3.1 *Karel*

Karel est à la fois un environnement et un langage de programmation (Bergin, J. Roberts et al., 1997). L'environnement Karel offre une interface graphique qui dispose d'un plan discret (dit *monde de Karel*) dans lequel évolue le robot programmable *Karel*. Les lignes horizontales sont appelées *Avenues* et les lignes verticales sont appelées *Streets* (Figure 18). Le robot peut se déplacer en ligne droite horizontale ou verticale, peut tourner et changer de direction, peut percevoir son environnement (e. g. détection de sons et d'obstacles) et interagir avec d'autre éléments présents dans son environnement (e. g. porter et déplacer des avertisseurs sonores) (Bergin, J. Roberts et al., 1997).

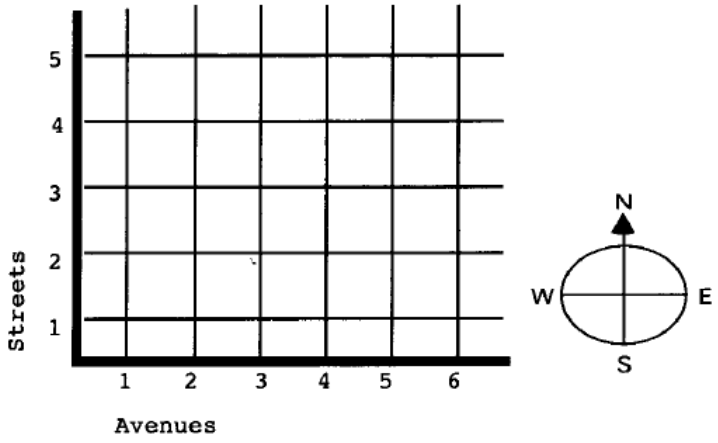


Figure 18 – Le monde du robot Karel (Bergin, J. Roberts et al., 1997).

Karel est un environnement Orienté-Objet, où le robot Karel est représentatif d'un objet informatique (Bergin, Stehlik et al., 2005). Il a connu plusieurs implémentations différentes reposant sur des langages différents. Karel++ est le premier langage utilisé dans sa version initiale (Bergin, J. Roberts et al., 1997). Une deuxième implémentation utilisait le langage de programmation Java, donnant lieu à l'environnement *Karel J Robot* (Bergin, Stehlik et al., 2005). Une troisième implémentation a conduit au développement de l'environnement *ObjectKarel* qui utilise le langage Karel++ (Xinogalos, Satratzemi et Dagdilelis, 2006) (Figure 19).

Le robot Karel est programmé au moyen de messages. Dans l'environnement Karel, programmer un robot revient à lui envoyer un message (Bergin, J. Roberts et al., 1997 ; Bergin, Stehlik et al., 2005). Plusieurs primitives d'instructions permettent de programmer le comportement général d'un robot, incluant des primitives de déplacement : `move` qui le fait déplacer d'une case dans le plan, `turnLeft` et `turnRight` qui le font tourner respectivement à gauche ou à droite. Karel dispose également d'une primitive de fin de tâche `turnOff` et de primitives d'interaction avec les avertisseurs sonores présents dans son environnement : `pickBeeper` et `putBeeper` qui ordonnent respectivement au robot de porter ou de déposer l'avertisseur sonore à l'origine du plan. L'ensemble de ces primitives désignées formellement par `méthodes` dans Karel J Robot constituent la

classe *UrRobot* décrivant le modèle générique d'un robot Karel. La déclaration de cette classe dans le langage Karel++ est faite comme suit (Bergin, J. Roberts et al., 1997) :

```
class ur_Robot{
     void move();
     void turnOff();
     void turnLeft();
     void turnRight();
     void pickBeeper();
     void putBeeper();
};
```

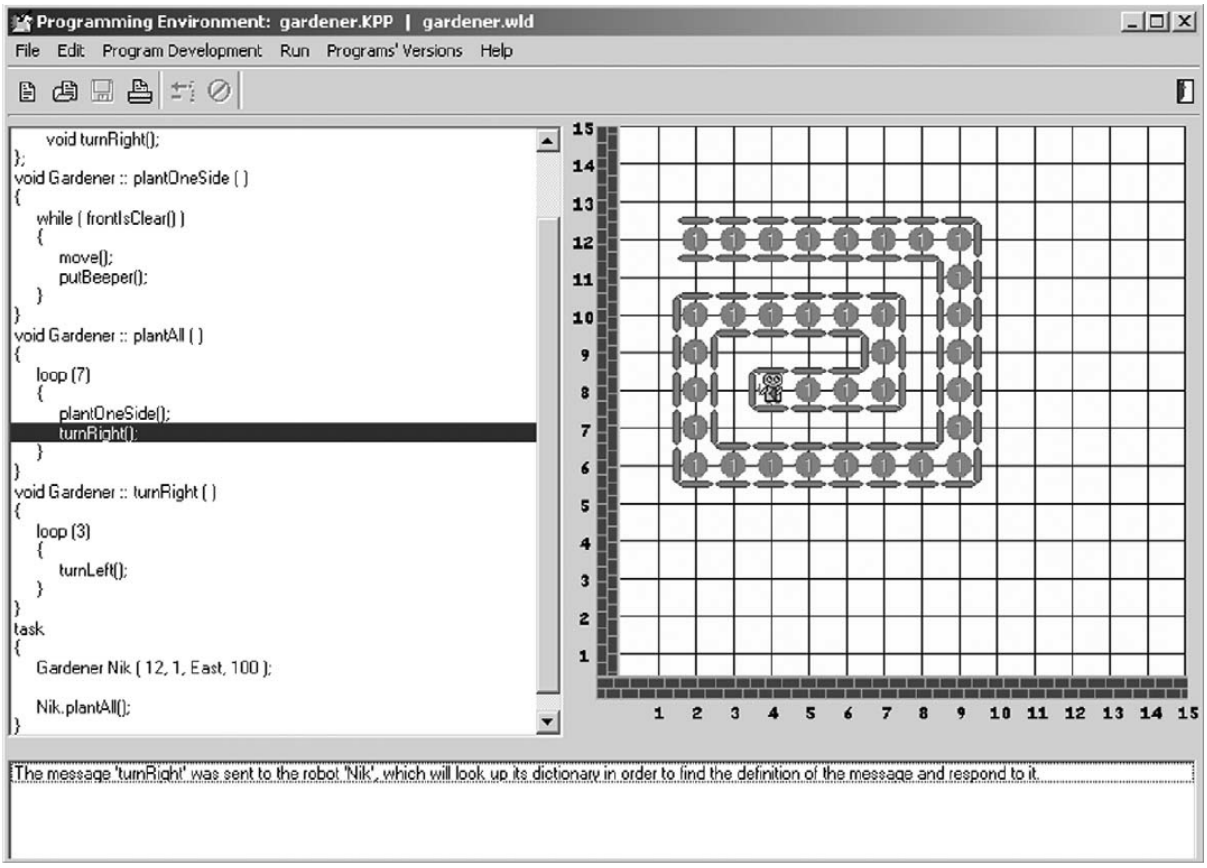


Figure 19 – L'environnement ObjectKarel (Xinogalos, Satratzemi et Dagdilelis, 2006).

Karel++ permettait également l'implémentation de structures de contrôle conditionnelles (`if (<test>) {<instructions>} else {<instructions>}`), d'itérations (`loop (<nombre positif>){<instructions>}` et `while (<test>) {<instructions>}`) et du polymorphisme en plus de l'héritage (Bergin, J. Roberts et al., 1997).

Au moyen d'héritage, la classe UrRobot peut être spécifiée, pour obtenir d'autres robots personnalisés comme dans l'exemple ci-dessous (Bergin, J. Roberts et al., 1997) :

```
class Robot: ur_Robot{
     Boolean frontIsClear();
     Boolean nextToABeeper();
     Boolean facingNorth();
};
```

Dans l'environnement *Karel J Robot*, les programmes sont saisis dans le langage Java. L'exemple ci-dessous montre la création d'un nouveau robot karel à la position (1,2) du plan se dirigeant vers l'est et ne portant pas d'avertisseurs sonores. Le robot doit ensuite se déplacer dans son monde afin de récupérer un avertisseur sonore puis le déposer (Bergin, Stehlik et al., 2005) :

```
public static void main(String [] args){
    UrRobot karel = new UrRobot(1,2,East,0);
    karel.move();
    karel.move();
    karel.pickBeeper();
    karel.move();
    karel.turnLeft();
    karel.move();
    karel.move();
    karel.putBeeper();
    karel.move();
    karel.turnOff();
}
```

L'extrait de programme ci-dessous implémente un exemple de polymorphisme : plusieurs robots se trouvent les uns à côté des autres. Chaque robot doit déposer un avertisseur sonore puis se déplacer en ordonnant au robot avoisinant d'exécuter la même tâche. La méthode `distributeBeepers()` de l'interface `BeeperPutter` sera définie différemment dans les classes `NoNeighbor` et `NeighborTalker` désignant respectivement un robot possédant un voisin et le dernier robot de la file, avant d'être utilisée dans le programme principale `main`.

```
public class NoNeighbor extends UrRobot implements BeeperPutter{
    public void distributeBeepers(){
        putBeeper();
        move();
    }
    <...>
}
public class NeighborTalker extends UrRobot implements BeeperPutter{
    private BeeperPutter neighbor = null;
    <...>
    public void distributeBeepers(){
        putBeeper();
        move();
        neighbor.distributeBeepers();
    }
}
public static void main(String [] args){
    BeeperPutter aRobot = new NoNeighbor(1,1,North,1);
    aRobot = new NeighborTalker (1,2,North,1,aRobot);
    aRobot = new NeighborTalker (1,3,North,1,aRobot);
    aRobot = new NeighborTalker (1,4,North,1,aRobot);
```

```
22      aRobot.distributeBeepers();
}
```

Karel a été utilisé en classe dans l'introduction de la POO aux débutants (Buck et Stucki, 2001 ; Xinogalos, Satratzemi et Dagdilelis, 2006) et a été présenté comme très simple à utiliser, grâce notamment à son interface qui incorpore un éditeur structuré offrant des menus de commandes et des boîtes de dialogues qui aident l'apprenant dans le développement de ses programmes.

3.3.3.2 *Alice*

Alice a été initialement conçu dans le but de simplifier la programmation 3D aux non-programmeurs (M. J. Conway, 1997 ; M. Conway et al., 2000). L'environnement Alice offre une API et une bibliothèque de modèles 3D permettant de peupler un environnement 3D restreint (`le monde Alice`) avec des Objets 3D choisis dans la bibliothèque de modèles, puis de programmer leur comportement en utilisant l'API Alice, qui était fondée sur une version légèrement modifiée du langage Python (M. J. Conway, 1997).

L'interface Alice était munie d'une boîte de commandes à une ligne de script, permettant de saisir et d'exécuter instantanément une commande sur un objet 3D présent dans la scène 3D (M. J. Conway, 1997). Par ailleurs, un objet 3D peut être manipulé directement dans la scène via un menu de commandes. Un onglet de script permettait également la saisie de scripts de plusieurs lignes de commandes dans le langage Alice. L'exécution du script produit une animation à l'écran. L'exemple ci-dessous montre un script dans lequel est programmé le comportement d'un personnage 3D nommé `bunny`. L'objet 3D doit reculer, faire un saut, retomber, puis rebondir de nouveau (M. J. Conway, 1997) :

```
doinorder (
2           #reculer
            bunny.resize(toptobottom,0.5,likerubber),
            #sauter
            dotogether(
               bunny.resize(toptobottom,2,likerubber),
7              bunny.move(up,1)
            ),
            #tomber
            bunny.move(down,1),
            #rebondir
12          bunny.resize(toptobottom,0.5,likerubber),
            bunny.resize(toptobottom,2,likerubber)
)
```

Alice a été ensuite introduit comme un environnement d'apprentissage de la programmation par Cooper, Dann et Pausch (2000), qui ont remarqué son potentiel dans l'introduction de la programmation aux débutants. Plusieurs versions de l'environnement ont été ensuite développées, dont notamment, Alice 2 (Figure 20) et Alice 3 [5].

5. Alice 2 et Alice 3 sont développés à l'Université Carnegie Mellon (États-Unis). http://www.alice.org

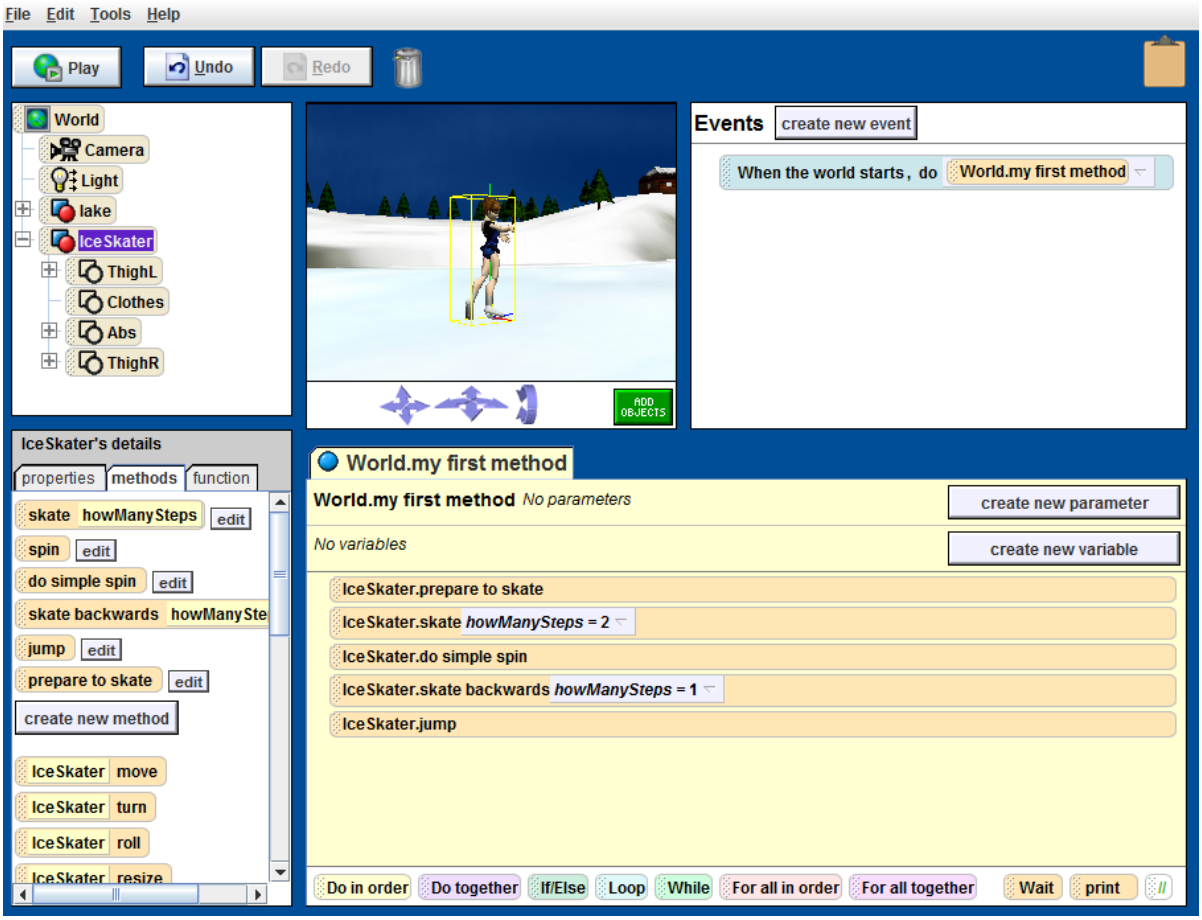


FIGURE 20 – L'environnement Alice 2 montrant un exemple de script agissant sur un personnage 3D.

Alice est conçu comme un environnement Orienté-Objet, où chaque objet 3D dans la scène est une représentation d'un objet informatique. Alice ne permet pas la modélisation 3D, mais possède une base de données de modèles 3D préalablement modélisés avec des outils dédiés. Cette base de données contient des fichiers définissant la géométrie des modèles, leur noms ainsi que la hiérarchie d'objets constituant chaque modèle. Ainsi quand un objet 3D est inséré dans la scène, un objet Python est créé et nommé conformément aux informations contenues dans le fichier le décrivant dans la base de données 3D. Les objets constituant ses différentes parties sont également créés et sont manipulables par le programmeur comme des attributs de l'objet les contenant.

L'API Alice définit un ensemble de classes permettant la programmation des animations. La première est la classe `Active` qui contient les méthodes élémentaires d'animations (e. g. `stop, start, pause, resume`). La classe `Hierarchical` définit les méthodes permettant la manipulation d'une hiérarchie d'objets (e. g. `BecomeChildOf, BecomeParentOf`). Deux autres classes `ACamera` et `ALight` définissent respectivement les méthodes permettant la gestion des caméras et des lumières présentes dans la scène 3D. La classe `AScene` implémente le nœud scène auxquels tous les autres objets 3D se rattachent. La classe `AnObject` contient la plupart des fonctionnalités de l'API qui constituent les commandes élémentaires les plus utilisées par un programmeur. Elle

regroupe des méthodes de manipulation d'objets (e. g. `move, moveTo, turn, turnTo, pointAt, setSize`), de requêtes de propriétés d'objets (e. g. `getPosition, distanceTo, getAngles, getScale`), de gestion de rendu (e. g. `get/setColor, get/setVisibility`), de gestion de texture (e. g. `setTexture`), etc. Ainsi, chaque objet 3D est une instance de la classe `AnObject`. Les détails de cette hiérarchie ne sont pas rendues explicites à l'utilisateur. Un programmeur utilise des fonctionnalités sans nécessairement savoir à quelle classe elles appartiennent.

Alice 2 est muni d'un éditeur de scripts glisser/déposer et repose entièrement sur le langage Alice. Le développement de scripts se fait sur la base des méthodes élémentaires disponibles dans l'API Alice. Lors de l'ajout d'objets dans la scène 3D, Alice invite l'utilisateur à créer une instance de classe afin de l'initier aux concepts Orientés-Objets. Une fois créé, l'objet est placé dans la scène 3D, la hiérarchie d'objets le constituant s'affiche dans l'onglet haut gauche de l'interface. Les méthodes élémentaire de sa classe s'ajoutent à l'onglet *Methods* en bas à gauche de l'interface, invitant l'utilisateur à constituer des scripts afin de créer des animations et des scénarios de jeux.

Alice 3 est est une extension de Alice 2, offrant d'avantage de fonctionnalités, notamment la possibilité de développer des scripts en langage Java et de les transférer dans un EDI Java (Dann et al., 2012).

Aujourd'hui, Alice est en continuelle évolution. Des cours d'initiation au langage Java et à la POO avec cet environnement sont proposés dans le programme de formation *Oracle Academy*[6].

3.3.3.3 *Greenfoot*

Greenfoot[7] est à la fois un environnement de développement intégré et un environnement éducatif pour l'apprentissage et l'enseignement de la programmation (Kölling, 2010a,b). Il est destiné à des apprenants âgés d'au moins 14 ans, tout en s'adaptant à des étudiants universitaires (Kölling, 2010b). Il permet de programmer le comportement de graphiques 2D pour la création d'animations et de scénarios de jeux. La programmation se fait dans le langage Java en utilisant l'API Greenfoot qui est entièrement fondée sur Java.

Greenfoot est un environnement Orienté-Objet. Son interface comporte deux parties essentielles : une scène 2D (appelée *monde*) dans laquelle s'exécutent les animations et un diagramme de classes des objets constituant le scénario courant (Figure 21). Il dispose également d'un éditeur pour la saisie de programmes Java et d'une bibliothèque d'images 2D pour la personnalisation de personnages lors de la création de classes.

Greenfoot introduit les concepts d'objet, de classe, d'héritage, d'appels de méthodes et de paramètres. Chaque objet du *monde* est représentatif d'un objet informatique, instance d'une classe visualisée dans le diagramme de classes. La Figure 21 montre un exemple de scénario utilisant les classes `World, TurtleWorld, Actor, Turtle, Lettuce et Snake`. Initialement Greenfoot visualise les classes `World et Actor` définies dans son API. La classe `World` peut être dérivée en choisissant l'image d'arrière-plan de la scène. Tous les autres objets ajoutés à la scène dérivent de la classe `Actor`, qui définit

6. https://academy.oracle.com/en/training-summary.html
7. Greenfoot est disponible sur le site : http://www.greenfoot.org

un certain nombre de méthodes permettant de programmer leur comportement dans l'éditeur, notamment la méthodes `act()` qui est appelée à chaque fois qu'une animation est exécutée. L'utilisateur peut ajouter des classes en spécialisant la classe `Actor` et en choisissant une image 2D pour personnaliser la nouvelle classe. La création d'un objet revient à sélectionner la classe à instancier dans le diagramme de classes, puis à le déposer dans la scène 2D. Une fois l'objet placé dans le monde, l'utilisateur peut avoir directement accès à ses propriétés pour interagir avec lui (Figure 22). Les méthodes de la classe `Actor` sont visibles, peuvent être directement appelées et le résultat de leur exécution est instantanément visualisé à l'écran.

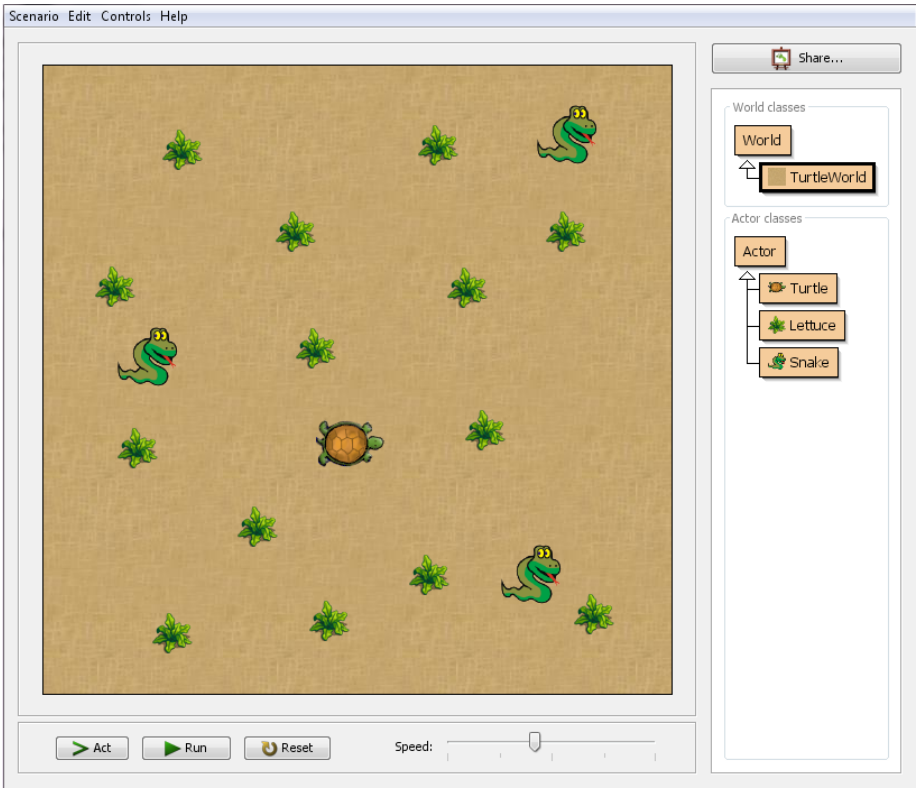
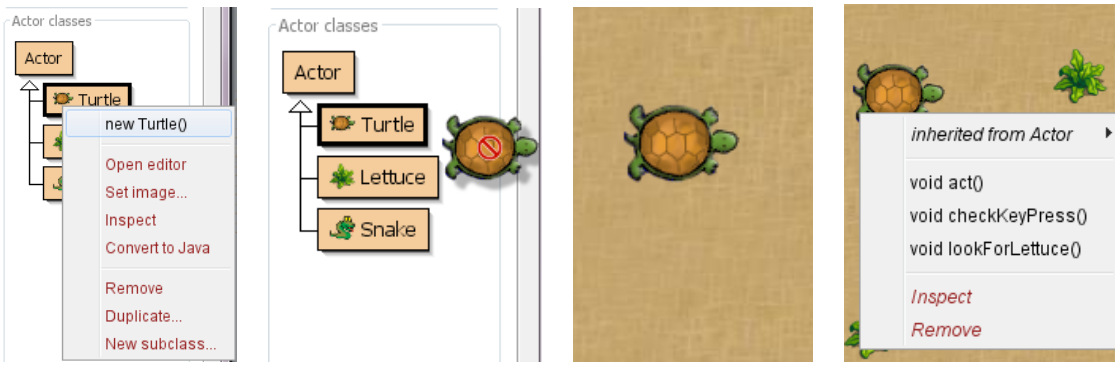


FIGURE 21 – Interface de Greenfoot montrant une scène 2D peuplée avec des personnages.

Le comportement de chaque objet peut être spécifié en modifiant sa classe dans l'éditeur de code. À chaque fois qu'une classe est ajoutée dans le diagramme de classes, son code source est créé automatiquement en Java et peut être modifié par l'ajout de nouvelles méthodes qui doivent être appelées dans la méthode `act()`. La création de nouvelle méthodes se fait grâce aux méthodes définies dans l'API Greenfoot.

L'API Greenfoot définit sept classes différentes permettant la programmation d'animations et de jeux : la classe `Greenfoot` fournit des méthodes pour gérer l'interaction avec le système. Les classes `GreenfootImage` et `GreenfootSound` permettent de gérer respectivement les images et les sons lors de la constitution de scénarios. La classe `MouseInfo` gère les événements souris. La classe `UserInfo` permet la sauvegarde de données utilisateur pour le partage de projets sur le web. Les classes `Actor et World` sont les classes visualisées dans l'interface et sont les plus utilisées par le programmeur. La classe `Actor` est la classe de tout objet présent dans le monde, elle définit toutes les méthodes relatives à la programmation du



(a) Instanciation de classe. (b) Visualisation de l'objet créé. (c) Placement de l'objet dans le monde. (d) Invocation de méthodes sur l'objet.

FIGURE 22 – Introduction aux concepts de la classe et d'objet dans Greenfoot.

comportement d'un objet (e. g. `act()`, `move()`, `turn()`, `isAtEdge()`, `isTouching()`, `getIntersectingObject()`, `getNeighbours()`, etc). La classe `World` est la classe du monde auquel sont rattachés tous les autres objets. Elle contient plusieurs méthodes, notamment `addObject()` qui ajoute un objet et le place dans la scène.

Le programme suivant montre un scénario constitué à partir des classes `Turtle` (classe de l'objet tortue) et `Lettuce` (classe de l'objet laitue) dérivant de la classe `Actor`. La tortue doit avancer à la recherche de laitues. À chaque fois qu'une laitue est trouvée par la tortue, le programme doit masquer la laitue trouvée, émettre un son et afficher le nombre de laitues que la tortue a trouvé sur son chemin. Pour ce faire, la méthode `lookForLettuce()` est ajoutée à la classe `Turtle`. Elle fait appel aux méthodes fournies dans la classe `Actor (isTouching(), removeTouching(),getWorld(), showText()` et `playSound())`. Cette méthode est ensuite appelée à l'intérieur de la méthode `act()` afin d'être exécutée dans l'animation.

```
1  public class Turtle extends Actor{
     private int lettucesEaten;
     public Turtle(){
          lettucesEaten = 0;
     }
6    public void act(){
           move(5);
           lookForLettuce();
     }
     public void lookForLettuce(){
11         if (isTouching(Lettuce.class)) {
               removeTouching(Lettuce.class);
               lettucesEaten = lettucesEaten + 1;
               getWorld().showText("Lettuces: " + lettucesEaten, 100, 30);
               Greenfoot.playSound("slurp.wav");
16         }
     }
   }
```

Greenfoot est en continuelle évolution. Le programme de formation Oracle Academy [8] le propose comme environnement d'initiation à la POO en Java.

Nous allons maintenant décrire le concept de micromondes de programmation, qui réunit l'ensemble des environnements de programmation visuelle et d'interfaces de programmation tangible. Nous allons tenter de définir ce concept ainsi que ses fondements conceptuels.

3.4 LES MICROMONDES

3.4.1 *Définitions et genèse*

Plusieurs auteurs ont contribué à la définition de la notion de micromonde (Bruillard, 1997 ; Hogle, 1995 ; Lawler, 1987 ; Papert, 1980 ; L. P. Rieber, 1996 ; Thompson, 1988). En particulier, la genèse des micromondes est décrite dans le livre de Lawler et Yazdani (1987) : "*Artificial Intelligence and Education : Learning environments and tutoring systems*", où il est rapporté que l'origine du mot *micromonde* remonte aux travaux de Winograd (1972), portant sur un programme de reconnaissance de langue anglaise qui instruit un robot appelé *SHRDLU*, opérant dans un domaine miniature en manipulant des blocs physiques sur une table. Ce mot a été ensuite introduit dans un rapport interne du MIT : "*The 72' progress report*" (Minsky et Papert, 1972), dans lequel le terme désignait une subdivision de domaines de résolution de problèmes en fragments plus petits, ainsi que les schèmes qui peuvent se former lors de l'interaction avec ces fragments. Le terme a par la suite été popularisé après l'apparition de l'environnement LOGO (cf. s. 3.2.2.1) et avec l'édition du livre de Papert (1980) :" *Mindstorms : Children, computers, and powerful ideas*", pour désigner *des domaines d'apprentissage devant être conçus comme génétiques*, en faisant référence à la genèse et à l'évolution de la connaissance, incluant ce qu'un individu peut apprendre et comment il peut l'apprendre. Ceci est fondé sur *l'épistémologie génétique*, un concept introduit par Piaget (1967) dont Papert (1980) a été le disciple. Papert (1980) soutient essentiellement l'idée que l'individu acquiert de la connaissance en la construisant lui-même.

Alors que Papert (1980) se concentre sur l'intention constructiviste des micromondes (cf. s. 3.4.4.4), Lawler (1987) décrit ces derniers comme des *mondes virtuels pour l'action créative : une idée essentielle est de créer un environnement avec lequel d'autres personnes pourront exercer leur créativité*. Cet environnement est constitué d'objets dits *transitionnels* (Lawler, 1987 ; Papert, 1980), car selon Lawler (1987), ces objets aident à développer de façon incrémentale des compétences impliquant la capacité à manipuler des objet formels. Les objets constituant un micromonde ont des propriétés communes à la fois avec les objets formels de la science et les objets plus concrets de l'expérience sensible ou du monde réel (Lawler, 1987 ; Papert, 1987). Papert (1987) les décrit comme des objets qui, d'une certaine façon, sont semblables à ceux avec lesquels on travaille dans le monde réel et, d'une autre façon, sont semblables à des objets abstraits : "*ce sont des objets qui aident à manipuler les objets abstraits*". Papert (1987) ajoute également que

8. https://academy.oracle.com/en/training-summary.html

cette capacité de créer des objets transitionnels, est une manière de combler l'écart entre *l'apprentissage intuitif* et *l'apprentissage formel*.

Thompson (1988) décrit un micromonde comme un système constitué d'objets, de relations entre objets et d'opérations transformant ces objets et relations, pouvant servir à créer d'autres objets. Ces objets sont visualisés graphiquement et constituent un modèle de concepts ou des concepts proposés à l'apprenant. Un micromonde donne forme à la structure d'un concept. La tâche de l'apprenant est d'intégrer mentalement cette structure afin de se l'approprier. Il constitue de ce fait, le cœur d'un *système axiomatique intuitif*, dans lequel les apprenants peuvent définir de nouveaux objets et opérations et étudier interactivement leurs propriétés (Thompson, 1988).

Dans le même ordre d'idées, Bruillard (1997) réserve l'expression *micromonde transitionnel* pour désigner les premiers micromondes qui ont été conçus, dont notamment LOGO. Il décrit un micromonde comme un monde artificiel dans lequel on agit sur des objets, dont le comportement respecte certaines contraintes de fidélité et de cohérence. L'idée essentielle sous-jacente à cette définition, est *la fidélité et la consistance* d'un certain modèle intégré et exprimé dans le micromonde. En effet, Bruillard (1997) affirme qu'un micromonde tente d'établir un *lien sémantique fort* entre le formel (*ou l'abstrait*) et le réel (*simulé ou de référence*), en garantissant la conservation du sens grâce aux connaissances de l'apprenant sur le fonctionnement des objets réels. Il a proposé un schéma général des différents éléments constitutifs des micromondes transitionnels (Figure 23), dans lequel le monde réel ou de référence est relié au monde formel ou abstrait par un processus de *modélisation* ou *d'exemplification*. Les objets visibles à l'interface se comportent d'une manière cohérente avec le modèle formel sous-jacent et leur aspect externe, c'est-à-dire à l'interface, rappelle les objets du monde réel.

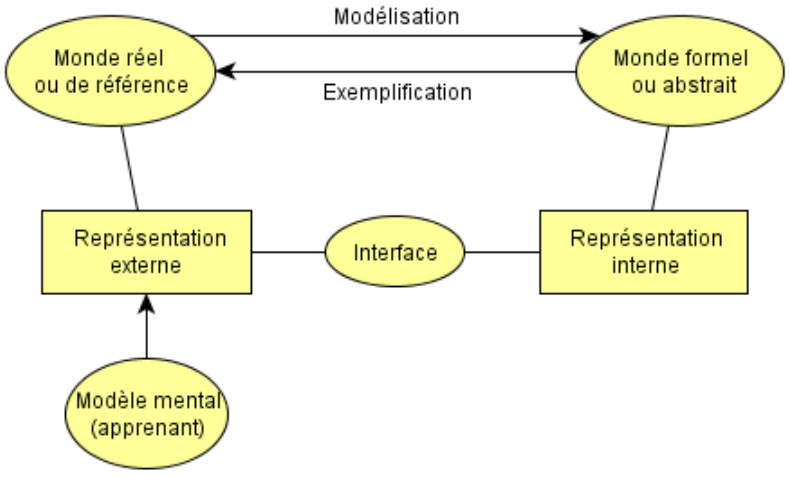


Figure 23 – Schéma général d'un micromonde transitionnel (Bruillard, 1997).

Ce schéma postule l'existence de modèles mentaux sur lesquels les micromondes sont fondés. Un modèle mental est vu comme une représentation dont les objets symboliques se comportent d'une manière similaire aux objets dans les situations où ils sont représentés (Bruillard, 1997). Ces modèles mentaux évoluent par expérience dans les environnement proposés (Bruillard, 1997 ; Hogle, 1995 ; L. P. Rieber, 1996). Les re-

cherches et les théories sur les modèles mentaux, suggèrent que *des personnes forment des modèles mentaux du monde réel dans l'intention de bien le comprendre et d'interagir avec* (L. P. Rieber, 1996), ce qui est en concordance avec l'approche constructiviste qui repose sur l'idée qu'un individu construit son apprentissage en interagissant avec son environnement (Papert, 1980). L. P. Rieber (1996) souligne trois attributs de modèles mentaux utiles à la conception de micromondes : *un système cible* qui est le système considéré contenant les objets de science. Le *modèle mental courant* du système cible de l'utilisateur, décrivant l'état courant des connaissances du système cible chez l'utilisateur. Enfin, un *modèle conceptuel du système cible* désignant l'artefact artificiel tel qu'il est conçu par l'ingénieur ou le pédagogue, dans le but d'aider l'utilisateur à comprendre le système cible. Par modèle conceptuel, L. P. Rieber (1996) entend ce qui est en partie approprié à l'utilisateur. Cela pourrait être, à titre d'exemple, une analogie que l'utilisateur connait. De ce fait un micromonde est vu comme un *modèle conceptuel interactif*. Ceci est en concordance avec les constituants du micromonde transitionnel vu par Bruillard (1997), où le système cible correspond au monde formel et le modèle conceptuel correspond à la fois au monde réel contenant les objets transitionnels, ainsi qu'à l'interface d'interaction du micromonde par laquelle l'utilisateur manipule ces objets. La construction du modèle mental est ici vue comme l'un des objectifs primaires des micromondes. La compréhension des connaissances ciblées est vue comme une évolution du modèle mental courant de l'apprenant.

L. P. Rieber (2005) a plus tard définit un micromonde comme un *environnement numérique interactif* qui permet à l'utilisateur d'interagir avec des *modèles de situations et de phénomènes*. Cet environnement génère un feedback dynamique basé sur un ensemble de règles sous-jacentes, modèles et opérations comme réponses aux manipulations de l'utilisateur des objets et paramètres constituant cet environnement. Cette définition est en lien avec celle proposée par Hogle (1995), qui décrit un micromonde comme un *environnement d'apprentissage interactif*, qui est un modèle conceptuel de certains aspects du monde réel. Il s'agit d'un environnement qui est souvent *idéalisé et simplifié*, dans lequel les apprenants explorent ou manipulent la logique, les règles ou les relations du concept modélisé, tel qu'il est défini par le concepteur. En ce sens, un micromonde est vu comme un moyen de simplifier et de concevoir des modèles du monde réel, permettant aux apprenants de manipuler et d'observer individuellement certaines contraintes et variables (Hogle, 1995).

3.4.2 *Rapport des micromondes aux simulations et aux jeux*

La recherche sur les micromondes comprend leurs rapports aux simulations et aux jeux (Hogle, 1995 ; L. P. Rieber, 1994). Ces trois types d'environnements peuvent partager des caractéristiques communes, mais restent mutuellement exclusifs. Tout dépend de leur conception et de leur utilisation dans un contexte d'apprentissage (L. P. Rieber, 1994).

Un micromonde se distingue d'abord des simulations par deux caractéristiques majeures (L. P. Rieber, 1994) : premièrement, un micromonde détient le modèle le plus simple d'un système ou d'un domaine qui reste reconnaissable par un expert de ce domaine. Deuxièmement, les paramètres d'un micromondes sont soigneusement conçus

pour correspondre au niveau, à l'expérience et à l'intérêt de l'apprenant. Cette seconde caractéristique est la plus importante, étant donné qu'elle offre à l'utilisateur, un point d'entrée au domaine considéré. En revanche, une simulation tend à imiter, d'une certaine manière un environnement ou un système réel ou imaginaire (L. P. Rieber, 1994). Les simulations tirent profit des techniques de visualisation et d'interaction de la Réalité Virtuelle (RV) qui transportent l'utilisateur d'une réalité à une autre. L'une des caractéristiques essentielles des simulations est l'existence d'un ensemble de règles et de modèles sur lesquels elles sont fondées (Willis, L. Hovey et K. G. Hovey, 1987). Les simulations possèdent deux objectifs (L. P. Rieber, 1994) : le premier est de fournir un moyen d'étudier un système particulier. C'est le cas des simulations scientifiques, à titre d'exemple, un météorologiste peut concevoir une simulation d'une tornade pour mieux comprendre les conditions sous lesquelles les tornades se forment. Dans ce cas, la simulation serait nécessairement basée sur une théorie du système. En d'autres termes, la simulation cherche à modéliser une théorie. De cette manière, les scientifiques peuvent vérifier et réviser leurs théories de phénomènes complexes, car une expérience directe est impossible, coûteuse ou risquée. Le second objectif des simulations est éducatif (Reigeluth et E. Schwartz, 1989). Tout comme les simulations scientifiques, les simulations éducatives sont utilisées quand les utilisateurs ne peuvent pas expérimenter le système directement, en raison de coût, de risque ou d'inaccessibilité. Les apprenants apprennent sur le système en observant les résultats de leurs actions ou décisions à travers un feedback généré par la simulation.

L. P. Rieber (1994) décrit le degré de réalisme dans une simulation comme sa *fidélité*. Il rapporte que la relation entre l'apprentissage et la fidélité dans une simulation n'est pas linéaire, mais elle dépend du niveau de compétences de l'apprenant. Le fait d'augmenter la fidélité dans une simulation ne va pas nécessairement augmenter l'apprentissage. Un haut niveau de réalisme dans une simulation pourrait être approprié aux experts, mais fatal aux novices.

L. P. Rieber (1994) tente de distinguer les micromondes des simulations, par le fait qu'une simulation peut être conçue de sorte à ce qu'elle ne présente aucune différence d'une expérience réelle, tels que les simulateurs de vols par exemple. Ce type de simulations ne peuvent pas être considérées comme des micromondes, car elles représentent autant de variables et de facteurs de l'expérience réelle que possible : *une simulation n'est pas un micromonde car elle n'est pas conçue pour s'adapter à l'utilisateur. C'est au contraire l'utilisateur qui doit s'adapter à la simulation. Une simulation peut devenir un micromonde, si elle est conçue de façon à permettre à l'apprenant novice de comprendre le modèle de connaissances sous-jacent* (L. P. Rieber, 1994).

Les micromondes sont distincts des jeux, mais incorporent des éléments de jeux afin de tirer profit de leurs propriétés qui impactent positivement l'apprentissage, dont notamment *les fonctions cognitives et la motivation intrinsèque* (L. P. Rieber, 1994, 1996).

L. P. Rieber (1996) explique que le jeu-game présente une fonction organisationnelle basée sur des facteurs cognitifs liés au jeu-play. Le jeu permet d'une part la mise en place d'une situation de *play* qui a pour résultat *l'assimilation* et, d'une autre part, l'imitation qui permet *l'accommodation* (L. P. Rieber, 1996). La situation de *play* et d'imitation constituent des stratégies *d'apprentissage naturel* à la fois chez les enfants et les adultes. Le jeu implique également la résolution de problèmes.

Les micromondes offrent souvent la possibilité de concevoir des scénarios de jeux imaginatifs. C'est le principe de l'apprentissage dit *par conception* (Perkins, 1986) ou *par construction* (Harel et Papert, 1991), qui constituent d'excellentes stratégies pour l'exploration d'un domaine de manière riche et significative (L. P. Rieber, 1996).

Le jeu représente également un outil d'apprentissage pouvant impacter la motivation intrinsèque de l'apprenant (Garris, Ahlers et Driskell, 2002). En effet, plusieurs recherches ont lié la question de la motivation intrinsèque à de nombreuses caractéristiques présentes dans le jeu tels que la narration, le fantastique, le feedback, etc. (cf. c. 1) (Dondlinger, 2007 ; Garris, Ahlers et Driskell, 2002 ; Malone, 1981). Ces caractéristiques sont alors souvent soigneusement incorporées dans les micromondes afin d'atteindre l'objectif de la motivation intrinsèque (L. P. Rieber, 1996).

3.4.3 *Rapport des micromondes à la programmation visuelle et tangible*

La programmation tangible et/ou visuelle fait partie intégrante des premiers micromondes qui ont été conçus, dont notamment LOGO. Papert (1980) explique que l'idée de la programmation dans l'environnement LOGO, est introduite à travers la métaphore *d'instruction de la tortue en lui apprenant de nouveaux mots*. Les utilisateurs du système LOGO commencent leur expérience de programmation en programmant la tortue de sorte à ce qu'elle réponde à de nouvelles commandes. Papert (1980) observa que la programmation en LOGO a permis aux élèves de comprendre de façon précise un certain nombre de concepts mathématiques et physiques, car ils sont amenés à comprendre leurs propres processus de pensée en verbalisant leurs expériences. Papert (1980) explique également que les élèves n'apprennent pas que des connaissances *par la programmation*, mais apprennent également *la programmation*. Plus encore, l'utilisation de LOGO ne peut se passer de l'apprentissage de la programmation. Une connaissance minimale de la programmation est indispensable pour savoir utiliser l'outil, comprendre son but, interpréter les erreurs et savoir les corriger (Papert, 1993). Un compromis est à trouver entre l'apprentissage de l'outil et les services qu'il peut rendre. Mais ce qui était un inconvénient est vite devenu un avantage. En effet, l'apprentissage par la programmation peut s'accompagner ou engendrer de nouveaux savoirs et savoirs-faire en programmation (Bruillard, 1997). Comme le souligne Papert (1993) : *l'apprentissage des aspects techniques de la programmation était devenu l'objet principale du projet LOGO, est les élèves ont beaucoup appris sur la programmation*. Ce qui était initialement conçu comme outil *d'apprentissage par la programmation* est devenu un outil *d'apprentissage de la programmation*. C'est aussi le cas de l'environnement Alice qui a été initialement conçu dans le but de simplifier la programmation 3D aux non-programmeurs (M. J. Conway, 1997) (cf. s. 3.3.3.2). Il a été ensuite exploité comme support d'apprentissage de la programmation (Cooper, Dann et Pausch, 2000).

En dehors de LOGO, d'autres environnements utilisant le langage LOGO, ou se basant sur le même principe que LOGO, ont été présentés comme de bons supports à l'apprentissage de la programmation (Cooper, Dann et Pausch, 2000 ; Perlman, 1976 ; Raffle, 2008). Leur objectif consiste à rendre la programmation plus accessible aux débutants, en leur permettant de programmer et de manipuler des représentations visuelles ou tangibles (souvent animées) des concepts de la programmation. Les micro-

mondes incluent les environnements de programmation visuelle et les interfaces de programmation tangible. C'est typiquement le cas de la machine à slots de Perlman (1976) que Lawler (1987) désigne explicitement comme un micromonde, reposant sur des commandes symboliques et tangibles du langage LOGO (cf. s. 3.2.2.2). Le système Topobo (cf. s. 3.2.2.3), bien qu'il n'ait pas été à notre connaissance désigné comme un micromonde, présente un grand nombre des caractéristiques de ce dernier. Son concepteur affirme qui l'a été conçu pour permettre à l'utilisateur d'effectuer des transitions entre leurs connaissances kinesthésiques et des connaissances de programmation (Raffle, Parkes et Ishii, 2004). Il souligne également certaines similarités avec LOGO : *Topobo permet d'introduire des idées abstraites, en offrant aux enfants la capacité de simuler les mouvements de leurs réalisations avec les mouvements de leurs propres corps* (Raffle, 2008). Les environnements Karel (cf. s. 3.3.3.1), Alice (cf. s. 3.3.3.2) et Greenfoot (cf. s. 3.3.3.3) ont été explicitement référencés comme des micromondes dédiés à l'apprentissage de la programmation et de la POO en particulier (Bennedsen, 2008 ; Cooper, Dann et Pausch, 2003 ; Henriksen et Kölling, 2004 ; Pears et al., 2007 ; Xinogalos, Satratzemi et Dagdilelis, 2006).

Dans ce qui suit, nous réservons l'expression "*micromondes de programmation*" pour désigner les micromondes ayant comme objectif primaire l'apprentissage de la programmation. La section suivante liste les caractéristiques de ce type d'environnements. Ce travail de caractérisation des micromondes de programmation, en particulier de POO, a fait l'objet d'une publication dans le "*Journal of Interactive Learning Research (JILR)*" (Djelil, A. Albouy-Kissi et al., 2016).

3.4.4 *Caractérisation des micromondes de programmation*

3.4.4.1 *Emploi de métaphores*

Les micromondes de programmation utilisent des métaphores dans le but d'approcher des concepts du monde réel, rendant ainsi les concepts de programmation plus faciles à appréhender par les débutants. En effet, Xinogalos, Satratzemi et Dagdilelis (2006) argumentent que cela permet de réduire la distance entre le modèle mental de l'apprenant et les concepts abstraits de la programmation. Dans de tels systèmes, l'expérience de programmation est vécue par l'apprenant comme un phénomène réel. Dans le contexte de la POO, l'objet est expérimenté comme un objet actif dans un monde graphique restreint. Le concept de classe est présenté comme une description de propriétés et de comportements de l'objet (e. g. Karel, Alice, Greenfoot).

Travers (1996) explique que l'informatique est elle-même une métaphore bien structurée et que les paradigmes de programmation sont fondés sur des métaphores. Ainsi, dans le paradigme OO, les objets informatiques sont représentés de façon métaphorique comme des objets *physiques et de société*. Par analogie aux objets *physiques*, les objets informatiques possèdent une apparence et un comportement. De la même manière, comme des objets *de société*, ils possèdent la capacité de communiquer entre eux (Travers, 1996).

La métaphore régissant l'OO représente de manière explicite une relation entre des structures informatiques et des objets du monde réel. Ceci est reflété dans les micromondes de POO existants (e. g. Karel, Alice et Greenfoot), qui visualisent les objets in-

formatiques comme ayant une existence physique avec les mêmes propriétés et comportements d'objets du monde réel, tout en étant capables de répondre aux actions de l'utilisateur, ce qui permet de les rendre concrets.

En informatique, la plupart des concepts sont fondés sur des métaphores, par conséquent des métaphores explicites sont souvent employées pour apprendre la programmation (MacConnell, 1993). Carroll et Mack (1999) rapportent que la plupart d'enseignants ont observé que fournir aux étudiants des analogies, peut les aider à apprendre. Ils ajoutent que les métaphores servent à codifier et à communiquer de nouvelles connaissances de façon compréhensible par des apprenants débutants (Carroll et Mack, 1999).

Dans le contexte de la programmation, le terme métaphore n'est pas employé au sens étroit du terme, mais au sens d'un *modèle métaphorique* riche et complexe (Travers, 1996). Ce modèle métaphorique est défini comme *un moyen de structurer des connaissances d'un domaine en les associant à des concepts et relations d'un autre domaine plus familier* (Lakoff et Johnson, 2008). En ce sens, la métaphore consiste en un système conceptuel structurant et constitue un moyen fondamental pour apprendre (Lakoff et Johnson, 2008).

Par ailleurs, étant donné que l'objectif des modèles métaphoriques est de décrire quelque chose de difficile à comprendre (*occulté*) en quelque chose d'intuitif et de facile à comprendre (*visible*), les métaphores sont souvent tirées du monde réel, permettant ainsi de rendre la programmation concrète et intuitive (Travers, 1996). Ceci est donc utile pour l'apprentissage de concepts difficiles à comprendre par les débutants.

(Mayer, 1975) a étudié l'impact des métaphores sur l'enseignement de la programmation. Il a observé que plusieurs concepts de programmation peuvent être appris plus facilement quand ils sont introduits à l'aide de métaphores concrètes. Plus tard, il a également observé que fournir aux étudiants débutants des modèles métaphoriques, a conduit à une amélioration de l'apprentissage par rapport à une présentation de concepts techniques plus littérale (Mayer, 1981). Par ailleurs, il a conclu que la familiarité des modèles aide à l'assimilation de contenus techniques en fournissant des contextes significatifs (Mayer, 1981).

L'animation est un autre aspect lié aux métaphores employées par les micromondes de programmation qui, d'après Travers (1996), offre un outil puissant pour l'analyse de paradigmes de programmation. Cela est dû notamment au fait que l'animation permet d'exploiter notre connaissance relative à l'action (Travers, 1996). Ce qui permet de penser en termes d'objectifs et de fonctions, conduisant à la compréhension d'un système sans connaître les détails de son implémentation (Travers, 1996).

3.4.4.2 *Emploi de visualisations*

L'une des caractéristiques conceptuelles des micromondes de programmation est la visualisation explicite de représentations de concepts de programmation dans des contextes réalistes et significatifs (e. g. Scratch, Karel, Alice et Greenfoot). L'apprenant n'entame pas son apprentissage en saisissant du code, mais commence à la place, par créer des objets graphiques. La manipulation directe de graphiques représentant des concepts OO peut aider les étudiants, comme dans la vie réelle, à penser en termes d'objets (Kölling, 2010b). En effet, la recherche a montré le potentiel de cette approche visuelle sur l'augmentation des performances d'apprenants débutants et son impact

sur l'enseignement de la programmation (Cooper, Dann et Pausch, 2003 ; Kelleher, Pausch et Kiesler, 2007 ; Moskal, Lurie et Cooper, 2004 ; Naps et al., 2002 ; Wang et al., 2009 ; Xinogalos, Satratzemi et Dagdilelis, 2006). De plus, il a été observé que les visualisations et l'interactivité dans un environnement d'apprentissage permettent l'expérimentation, suscitent la curiosité et augmentent l'engagement des apprenants (Henriksen et Kölling, 2004).

Par ailleurs, les représentations visuelles sont étroitement liées aux métaphores, qui sont couramment employées pour expliquer des concepts complexes. En effet, les métaphores visuelles s'avèrent plus performantes que les métaphores verbales (Baldwin et Kuljis, 2001).

Gilbert (2010) souligne l'importance de la visualisation dans la conduite de la science, en la définissant comme une construction mentale personnelle, conduisant à la mémorisation d'informations et à la construction de l'apprentissage. En informatique, les visualisations sont largement utilisées afin d'illustrer graphiquement divers concepts et processus (Esteves et Mendes, 2003 ; Milne et Rowe, 2004 ; Moreno et al., 2004 ; Osman et Elmusharaf, 2014 ; Sajaniemi, Byckling et Gerdt, 2007). Cette nécessité de la visualisation vient notamment de l'abstraction inhérente aux concepts. En effet, Jerding et Stasko (1994) expliquent que le paradigme de POO est lui-même une fondation naturelle pour la visualisation, étant donné qu'il entraîne la manipulation d'objets concrets : *instances, messages, héritage*, etc. Par conséquent, la constitution de visualisations pour les concepts OO découle naturellement de leur correspondance à des représentations visuelles.

3.4.4.3 *Incorporation d'éléments de jeux*

Une autre caractéristique majeure des micromondes de programmation, est l'incorporation d'éléments de jeu. En effet, le jeu est une implémentation courante des micromondes en général (Hoyles, Noss et Adamson, 2002 ; Plass et R. N. Schwartz, 2014 ; L. P. Rieber, 2005) et des micromondes de programmation en particulier (Cooper, Dann et Pausch, 2003 ; Kölling, 2010b). Ceci s'explique, notamment, par le fait que certains enseignants sont convaincus que le jeu peut aider les apprenants à construire leur apprentissage et augmenter leur intérêt pour la discipline de l'informatique (Tapia et al., 2007). Dans le contexte de la POO, les micromondes offrent aux apprenants la possibilité de concevoir et de développer leurs propres jeux. C'est typiquement le cas de l'environnements Alice (Cooper, Dann et Pausch, 2003) et Greenfoot (Kölling, 2010a), qui permettent la construction de scénarios de jeux interactifs par la programmation et la manipulation de concepts OO.

Le jeu est exploité afin d'atteindre deux objectifs majeurs des micromondes (cf. s. 3.4.1). Le premier est de permettre à l'apprenant de construire de façon autonome, par l'expérience et la découverte des concepts complexes et abstraits. Le deuxième est de fournir des outils d'apprentissage engageants, en lien avec l'intérêt des apprenants (Djelil, A. Albouy-Kissi et al., 2016).

Le premier objectif s'inscrit dans l'approche d'apprentissage constructiviste (cf. s. 3.4.4.4). En effet, cela a été confirmé par plusieurs auteurs : "*la théorie constructiviste peut être conduite par l'usage de jeux*" (Papert, 1980) ; "*en développant des jeux, les étudiants apprennent à programmer dans une approche constructiviste*" (Kafai, 2006).

Le second objectif des micromondes est en lien avec la proposition d'activités d'apprentissage qui sont intrinsèquement motivantes (Papert, 1980). En effet, Lawler (1987) explique que les instanciations concrètes de concepts computationnels qui peuvent être considérés comme des symboles, font des micromondes des outils engageants. Un individu peut être engagé avec des objets computationnels qu'il interprète comme des symboles d'objets réels, manipulables uniquement à l'aide d'un langage informatique. Cela le motive à apprendre un ensemble d'opérations qui transforment l'état d'un objet, conduisant à une expérience avec les détails surfaciques d'un système formel, dont il peut apprécier les propriétés.

Selon Cooper, Dann et Pausch (2003) et Kelleher, Pausch et Kiesler (2007), les étudiants s'engagent activement dans la découverte et la construction de concepts et de compétences, quand ils sont amenés à concevoir et à créer leurs propres jeux. Par conséquent, les jeux sont en mesure de fournir un environnement approprié permettant un apprentissage actif et une motivation intrinsèque.

3.4.4.4 *Apprentissage constructiviste*

Les micromondes sont fondés sur la théorie d'apprentissage constructiviste, comme l'indiquent plusieurs auteurs : "*les micromondes sont des environnements interactifs dans lesquels on peut apprendre sur un domaine spécifique, par la découverte et l'exploration*" (Papert, 1980) ; "*la puissance des micromondes comme outils d'apprentissage réside dans la théorie constructiviste* " (Bliss et Ogborn, 1989) ; "*les micromondes constituent une application immédiate de l'incorporation du constructivisme à la conception pédagogique*" (L. P. Rieber, 1992).

Le constructivisme postule qu'un individu apprend, lorsqu'il construit activement des idées et des relations entre idées dans son mental, en se basant sur des expériences qu'il fait (Nix et Spiro, 1990). En ce sens, le constructivisme consiste essentiellement en l'idée que l'apprentissage implique la construction de connaissances par l'individu lui-même. Selon Hogle (1995), les micromondes permettent d'aller plus loin dans ce sens, car ils offrent un feedback graphique et rapide permettant l'auto-correction, et donc ils sont en mesure d'aider l'apprenant à déterminer l'exactitude de ses solutions sans l'intervention d'un enseignant.

Les micromondes reposent sur le constructivisme, étant donné qu'ils emploient des métaphores visuelles qui aident l'apprenant à construire des connaissances et des compétences par eux même. En effet, comme l'indiquent Carroll et Mack (1999), la fonction primaire d'une métaphore est de stimuler l'apprentissage actif. En ce sens, l'utilisation de métaphores facilite l'apprentissage actif, en fournissant des clés permettant des inférences logiques, grâce auxquels les apprenants construisent leurs connaissances (Carroll et Mack, 1999).

Ceci est en concordance avec la proposition de Hadjerrouit (2005), concernant la définition d'un framework pédagogique fondé sur une approche constructiviste pour l'enseignement de la conception et de la programmation OO. Hadjerrouit (2005) suggère que les connaissances OO doivent être activement construites par les apprenants. Le développement de programmes doit être guidé par des concepts OO, non par des aspects techniques du langage de programmation. Les apprenants doivent se confronter à des activités se concentrant sur des problèmes réalistes et engageants. Les fondements

de ce framework correspondent aux caractéristiques des micromondes de programmation, qui sont fondés de manière intrinsèque sur le constructivisme.

3.5 Synthèse : quels artefacts pour l'initiation à la programmation par le jeu-play ?

3.5.1 *Programmation tangible et visuelle : un moyen de combler l'écart entre les connaissances intuitives et les connaissances formelles*

Depuis la fin des années 1960, l'enseignement de la programmation était une grande ambition de plusieurs chercheurs. L'objectif étant de rendre la programmation accessible à tous, dès le plus jeune âge, et de ne plus la réserver qu'aux professionnels. Il s'agissait de développer une démarche attrayante pouvant conduire à une compréhension fine des concepts de la programmation et de la faciliter. C'est dans ce contexte que la programmation tangible et la programmation visuelle se sont développées. La programmation tangible suppose soit :

- de visualiser le résultat du processus de programmation sur des objets physiques hors de l'ordinateur ; e.g. LOGO (Papert, 1980).
- de manipuler des objets physiques lors du processus de programmation, sans recourir à un langage de programmation, puis de visualiser le résultat dans l'ordinateur ; e.g. TORTIS (Perlman, 1976).
- de manipuler des objets physiques et de visualiser le résultat des actions de manipulation directe sur des objets physiques, le tout hors de l'ordinateur, sans recourir à un langage de programmation ; e.g. Topobo (Raffle, Parkes et Ishii, 2004).

De manière similaire à la programmation tangible, la programmation visuelle et la visualisation de programmes emploient des objets graphiques dans la manipulation des concepts de programmation. Il s'agit soit :

- d'élaborer des programmes de manière graphique ; e.g. Pygmalion (Smith, 1975).
- d'illustrer l'exécution des programmes de manière graphique ; e.g. Prog&Play (Muratet, 2010).
- d'élaborer des programmes et d'illustrer leur exécution de manière graphique ; e.g. Scratch (Resnick et al., 2009), Karel (Bergin, J. Roberts et al., 1997), Alice (Cooper, Dann et Pausch, 2000), Greenfoot (Kölling, 2010b).
- d'élaborer des programmes de manière graphique et de visualiser leur exécution sur des objets physiques hors de l'ordinateur ; e.g. LogoBlocks (Begel, 1996), Lego Mindstorms (Baum et Zurcher, 2003)

L'apprentissage des concepts de programmation par le visuel et le tangible permet de s'affranchir de la syntaxe d'un langage de programmation textuel et/ou de concrétiser des concepts abstraits de la programmation, en visualisant le résultat d'exécution de programmes sur des objets physiques ou graphiques. Les environnements de programmation visuelle et tangible sont présentés comme particulièrement adaptés pour aider les novices à apprendre la programmation (Clarisse et Chang, 1986 ; Papert, 1980). Les objets visuels ou tangibles donnent forme à des concepts abstraits qui deviennent concrets. Cela permet à un apprenant débutant d'acquérir des connaissances formelles, au travers de ses connaissances du monde réel ou de son expérience sensible. Cela per-

met la transition entre des connaissances intuitives et des connaissances formelles, d'où l'expression "*objets transitionnels*" utilisée par certains auteurs (Bruillard, 1997 ; Papert, 1980). C'est cette propriété des micromondes de programmation qui aide l'apprenant débutant à comprendre facilement les concepts formels et abstraits.

3.5.2 *Micromondes de programmation : des systèmes transitionnels et des espaces de créativité*

3.5.2.1 *Les micromondes de programmation : des systèmes de représentation transitionnels*

Les *objets transitionnels* sont des éléments constitutifs des micromondes de programmation. Il permettent à l'apprenant d'accéder à des concepts de programmation formels et abstraits, au travers d'entités visuelles et concrètes. Ces dernières sont étroitement liées à ce que l'apprenant connait de son expérience sensible (cf. s. 3.4.4). Les objets transitionnels sont choisis de manière à établir un lien sémantique fort entre des concepts formels (ou des objets de science) et des objets concrets du monde réel (ou de l'expérience sensible). Cela nous amène à montrer un nouveau concept qui est celui de *système de représentation transitionnel, pour faire référence au système de représentation propre aux micromondes de programmation, désignant les éléments constitutifs des micromondes de programmation*. Ce système de représentation est constitué d'objets transitionnels, visuels et interactifs directement manipulables par l'apprenant. Les objets transitionnels possèdent des propriétés semblables à celles des concepts formels de la programmation et leur aspect externe rappelle des objets du monde réel (Figure 24). L'interaction de l'apprenant avec ces objets transitionnels, lui permet de produire du sens sur les concepts de programmation formels, car aidé par leurs représentations concrètes et significatives, tirées du monde réel.

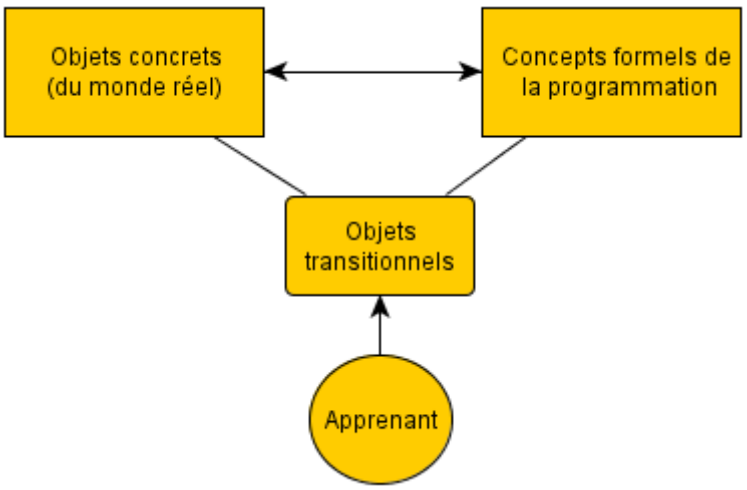


Figure 24 – Système de représentation transitionnel propre aux micromondes de programmation.

3.5.2.2 *Les micromondes de programmation : des espaces de créativité et d'imagination combinant l'apprentissage intuitif et le jeu-play*

Les micromondes de programmation sont conçus de façon à permettre *les actions de créativité*. Ils offrent des espaces dans lesquels l'apprenant est amené à exercer sa créativité et donner libre court à son imagination. Ce sont des environnements interactifs qui génèrent des feedbacks dynamiques basés sur un ensemble de règles sous-jacentes au modèle de connaissances formelles du paradigme de programmation implémenté. Ces feedbacks sont générés en réponses aux manipulations directes de l'apprenant sur les objets transitionnels et autres paramètres constituant le micromonde. Ces propriétés d'actions créatives et de feedback dynamique conduisent à la mise en place de situations de *jeu-play*. Cela fait des micromondes de programmation des outils à la fois puissants dans l'introduction des concepts abstraits, attrayants et engageants.

3.5.2.3 *Quatre dimensions conceptuelles à l'origine du potentiel des micromondes dans l'introduction de la programmation*

Les micromondes de programmation se fondent sur quatre dimensions conceptuelles liées entre elles, à savoir la *métaphore, la visualisation, le jeu-play et l'apprentissage (constructiviste)*(Figure 25).

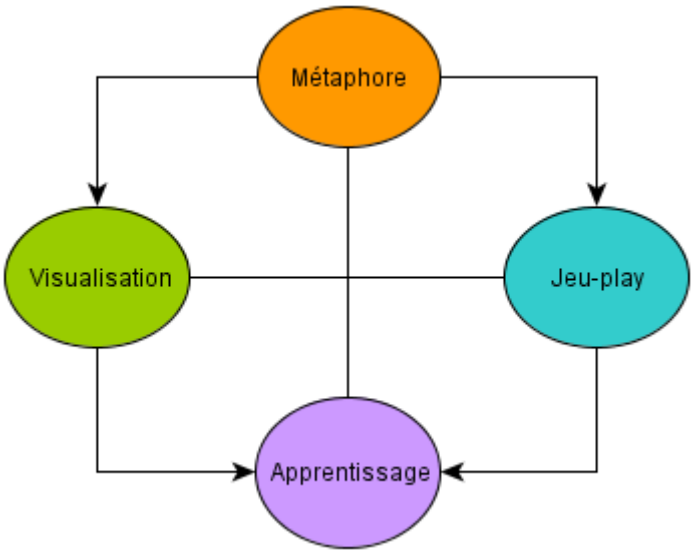


Figure 25 – Dimensions conceptuelles des micromondes de programmation.

Les objets transitionnels visibles à l'interface d'un micromonde de programmation relèvent d'une métaphore. L'objectif étant de réduire la distance avec les concepts formels de la programmation, et de l'OO en particulier. Dans un tel système, l'expérience de programmation est vécue par l'apprenant comme une expérience du monde réel. Les métaphores servent à décrire des concepts abstraits, donc difficiles à comprendre, en quelque chose de familier et d'intuitif, donc facile à comprendre. Cette propriété a été présentée comme utile à l'apprentissage de concepts, qui ne peuvent pas être directement perçus par les débutants et peut aider dans l'assimilation de contenus techniques

en fournissant des contextes significatifs (Carroll et Mack, 1999 ; Mayer, 1981 ; Travers, 1996). La métaphore est illustrée visuellement dans les objets transitionnels. La nécessité de visualisation vient justement de l'abstraction inhérente aux concepts de programmation. Le potentiel de métaphores visuelles dans l'apprentissage de la programmation a été démontré par plusieurs auteurs (Cooper, Dann et Pausch, 2000 ; Naps et al., 2002 ; Wang et al., 2009 ; Xinogalos, Satratzemi et Dagdilelis, 2006).

Le jeu-play est engendré notamment par l'activité de créativité de l'apprenant (conception de scénarios de jeux, animation de personnages, narrations, etc.) et la présence d'un feedback dynamique. Le jeu-play est aussi en lien avec la métaphore visuelle employée (personnages, animations, narrations, etc). La métaphore, la visualisation et le jeu-play conduisent à la construction active des connaissances chez l'apprenant, par l'adaptation, l'assimilation, la découverte et l'exploration. Il s'agit d'un fondement de l'apprentissage constructiviste, qui constitue une propriété essentielle des micromondes, en contribuant à leur potentiel dans l'apprentissage de la programmation (Hadjerrouit, 2005 ; Papert, 1980).

3.6 conclusions et implications

Ce chapitre nous a permis, d'une part, d'identifier les variables médiatrices associées à la conception d'un micromonde de programmation, et d'autre part, de comprendre son potentiel dans l'initiation des débutants à la programmation et à l'OO en particulier.

Les micromondes de programmation sont utilisés afin de favoriser la compréhension des concepts abstraits de la programmation. Ce sont des environnements interactifs et restreints, qui contiennent des modèles programmables d'expériences du monde réel. Ils sont engageants, attrayants et permettent un apprentissage fondé sur la participation active des apprenants, la construction individuelle de connaissances, la découverte et l'exploration dans des contextes significatifs et ludiques. Ils exploitent les connaissances intuitives de l'apprenant afin de lui permettre d'assimiler et de comprendre des connaissances formelles, au travers d'une métaphore et au moyen de graphiques ou d'objets physiques. Ils permettent la mise en place de situations de jeu-play, car d'une part, ils constituent des espaces de créativité et d'imagination, et d'autre part, ils offrent un feedback dynamique en réponse aux actions de l'apprenant.

Dans le contexte de la POO, ils s'étendent au-delà des langages de programmation, afin de permettre à l'apprenant d'explorer et de comprendre le modèle inhérent au paradigme OO à travers la manipulation directe d'entités graphiques à l'écran. À titre d'exemple, le concept d'objet est représenté graphiquement de sorte à ce que le résultat d'exécution d'une commande soit directement visible à l'interface, tels que sa position, sa taille, sa rotation et autres changements de son état. L'objectif d'une telle conception est de permettre à l'apprenant de penser l'objet informatique en terme de sa représentation graphique, conduisant à la construction d'un modèle de représentation de ce concept.

Le potentiel des micromondes dans l'introduction de la programmation aux débutants a été observé et démontré dans la littérature. Ce potentiel s'explique notamment par quatre dimensions conceptuelles qui sont liées entre elles, à savoir l'emploi de métaphores, l'emploi de visualisations, la mise en place de situations de jeu-play et la

construction individuelle de connaissances. La métaphore est illustrée par des visualisations qui permettent d'établir un lien sémantique fort avec des concepts formels et abstraits. Cela permet à l'apprenant de produire des connaissances formelles à partir de ses connaissances intuitives. Un micromonde de programmation est donc fondé sur un système de représentation transitionnel, qui est volontairement combiné avec des activités créatives et imaginatives conduisant à une situation de jeu-play. L'apprentissage découle alors des interactions de l'apprenant avec les objets transitionnels lors de la situation de jeu-play. L'apprenant construit ses connaissances de manière active grâce à ses interactions dans un contexte significatif, attrayant et engageant. C'est justement l'ensemble de ces propriétés que nous retenons dans la conception d'un nouveau micromonde de POO. Nous retenons également qu'il est primordial de choisir une métaphore pertinente qui peut aider l'apprenant à accéder facilement au modèle conceptuel régissant le paradigme OO, et donc d'arriver à un apprentissage efficace des fondamentaux de la POO.

PUBLICATIONS

Quelques idées et propositions décrites dans ce chapitre sont apparues dans les publications suivantes :

Djelil, Fahima, Adélaïde Albouy-Kissi, Benjamen Albouy-kissi, Eric Sanchez et Jean-Marc Lavest (2016). « Microworlds for learning Object-Oriented Programming : Considerations from research to practice ». In : *Joural of Interactive Learning Research* 27.3, p. 265–284.

Djelil, Fahima, Adélaïde Albouy-Kissi, Benjamin Albouy-Kissi, Eric Sanchez et Jean-Marc Lavest (2015). « Alice, Greenfoot et Prog&Play ou comment apprendre la programmation orientée-objet par le jeu ». In : *Environnements Informatiques pour l'Apprentissage Humain (EIAH)*, p. 420–422.

Deuxième partie

DÉMARCHE MÉTHODOLOGIQUE ET CONTRIBUTIONS

Cette partie décrit les méthodes employées dans la mise en œuvre de nos propositions concernant les principes de conception et d'évaluation d'un nouveau micromonde de Programmation Orientée-Objet, compatible avec une classe multi-dispositifs tactile. Cette partie détaille, dans un premier temps, un framework d'applications que nous avons conçu afin de prendre en compte les exigences de notre contexte de travail, la classe *Tactileo* (Chapitre 4). Nous présentons, ensuite, un nouveau micromonde de Programmation Orientée-Objet appelé PrOgO, dont l'objectif pédagogique est d'aider les étudiants débutants à développer des connaissances sur les concepts fondamentaux de la Programmation Orientée-Objet dans un contexte de jeu-play (Chapitre 5). PrOgO implémente les principes de conception issus de notre revue de la littérature décrite dans la première partie, et permet de mettre en expérience nos propositions d'évaluation des apprentissages par la génération de données sur les interactions des apprenants avec son interface. Enfin, cette partie s'achève par la description de notre méthodologie d'évaluation, qui met l'accent sur l'analyse des usages de PrOgO au travers de traces d'interaction numériques, et l'emploi de techniques d'analyse relevant du domaine de l'analyse de l'apprentissage (*Learning Analytics*) (Chapitre 6).

4

UNIFICATION D'APPLICATIONS POUR LA CLASSE *TACTILEO*

4.1 INTRODUCTION

Le besoin d'un framework d'applications multiplateformes pour interfaces multi-touches tangibles et pour la découverte automatique du réseau est né du projet *Tactileo*.

Au commencement du projet *Tactileo* à l'université d'Auvergne (IUT du Puy en Velay) en Octobre 2013, quand nous avons commencé la réflexion sur des applications conçues pour être utilisées dans la classe *Tactileo*, nous avons décidé de fournir au-delà d'applications spécifiques, un framework d'applications permettant à un développeur informaticien de facilement relier des dispositifs multiples, incluant des interfaces utilisateurs multi-touches et tangibles, et de construire diverses et riches applications éducatives. L'objectif de ce travail entrepris au début de ce projet, n'est pas de soulever des questions de recherche relevant du domaine de l'IHM, et d'étudier l'intégration d'interfaces tactiles hétérogènes dans un environnement de classe. Notre objectif est plus modeste, il s'agit de concevoir un framework d'applications qui sont destinées à être utilisées dans la classe *Tactileo*. Le framework fournit une couche d'abstraction pour le développement d'applications multiplateformes, une interface logicielle unifiée pour la gestion d'évènements utilisateurs tactiles et tangibles, et une interface réseau ayant pour objectif la simplification de la communication réseau, par la découverte automatique, entre plusieurs applications fonctionnant sur de multiples dispositifs hétérogènes.

Le présent chapitre est structuré comme suit. La nécessité d'un framework d'applications pour la classe *Tactileo* est d'abord justifiée (Section 4.2), avant de décrire le framework *Tactileo* (Section 4.3). Nous décrirons son architecture ainsi que ses exigences techniques. Nous présenterons ensuite, les protocoles implémentés dans le développement du framework, ainsi que les détails de sa conception et de son implémentation. L'objectif, ici, n'est pas de présenter de manière exhaustive les différents protocoles employés, mais de simplement présenter leurs différentes fonctionnalités exploitées dans le développement du framework, et la manière dont ils sont implémentés au sein du framework. Nous présenterons également comment le framework peut être exploité pour le développement d'applications, avant de souligner quelques-unes de ses forces et limites (Section 4.4). Enfin, nous conclurons par nos contributions (Section 4.5).

4.2 NÉCESSITÉ D'UN FRAMEWORK D'APPLICATIONS

Les applications éducatives conçues en vue d'être utilisées dans des environnements de classes multi-dispositifs, possèdent des besoins larges et divers en termes d'exploitation d'interfaces utilisateurs interactives.

Dans le cas d'environnements de classes multi-tables, plusieurs projets ont essayé d'étudier l'interaction entre la table de l'enseignant et les tables d'élèves, dans le développement d'applications adaptées aux besoins de l'enseignant et des élèves. À titre

d'exemple, dans le cadre du projet *SynergyNet* (Burd et al., 2013), une technique d'interaction appelée *TablePortal* (AlAgha et al., 2010) a été développée afin de faciliter la communication entre la table de l'enseignant et les tables d'élèves dans la classe. Cette technique fournit un "*portail*" de communication à une table distante connectée à un même réseau, permettant à l'utilisateur de voir et d'interagir avec son contenu. Ce système permet à l'enseignant d'interagir avec du contenu sur des tables d'élèves distantes. Il permet la diffusion de ressources éducatives et la supervision d'activités d'élèves. Il permet également à l'enseignant d'intervenir dans un travail de groupe spécifique et de déplacer du contenu de surfaces horizontales vers des surfaces verticales. *MTClassroom* (Martinez-Maldonado et al., 2013) est un autre exemple de classes multi-tables, qui capte les interactions d'élèves travaillant en petits groupes. Un outil appelé *MTDashboard* est fournit à l'enseignant sur un dispositif portable, afin de superviser les activités de classe, affichant en temps réel, des indicateurs de participation et de progression des tâches de chaque groupe.

Dans le contexte d'environnements multi-dispositifs, on peut citer le système *MultiSpace* (Everitt et al., 2006), qui regroupe plusieurs dispositifs tels que des tables, des murs interactifs, et des ordinateurs pour le transfert de documents. C'est un système *centré-table*, au sein duquel l'interaction avec de multiples dispositifs distants est réalisée depuis une table interactive. Le mur est relié à la table via un réseau filaire, et les ordinateurs sont connectés à la table via un réseau local sans fil. Le système *MultiSpace* permet à l'utilisateur de transférer des objets depuis une table centrale vers des dispositifs distants interconnectés, en les glissant dans une fenêtre *portail* s'affichant sur un coin de la table. *ARIS* est un autre exemple (Biehl et Bailey, 2004), qui est un espace interactif pour la gestion de fenêtres, permettant à l'utilisateur de facilement gérer des informations au travers d'assistants numériques personnels (PDAs), d'ordinateurs, de tablettes, d'écrans larges interactifs et d'autres dispositifs informatiques. ARIS fournit une interface de manipulation directe montrant un plan iconique de l'espace de travail courant, permettant aux utilisateurs de réallouer visuellement des applications et de rediriger des entrées au travers de dispositifs multiples.

Ce bref aperçu montre l'existence de quelques systèmes de gestion d'applications dédiées aux classes multi-dispositifs, incluant des interfaces tactiles multi-touches offrant des fonctionnalités très riches. Cependant, au commencement de notre projet, il n'existait à notre connaissance, aucune interface logicielle conçue pour être portable et facile à intégrer dans le développent de nouvelles applications, tout en prenant en compte les besoins d'environnements multi-dispositifs, comprenant des tables multi-touches tangibles, des tablettes, des ordinateurs et autres dispositifs informatiques. Il paraissait plutôt que chaque système construit sa propre interface logicielle afin de réaliser des objectifs bien définis. Par ailleurs, on connaissait l'existence de quelques frameworks logiciels ouverts (open source) pour le développement de nouvelles applications dédiées aux interfaces multi-touches et aux interfaces tangibles. Cependant, ces frameworks ne considèrent pas les applications dédiées aux classes multi-dispositifs. On peut citer le framework *SynergySpace* (AlAgha et al., 2010), qui est un framework logiciel ouvert qui permet le développement rapide d'applications multi-touches et l'intégration de diverses fonctionnalités de mise en réseau, afin de construire des systèmes dédiés uniquement aux tables tactiles multi-touches. D'autres types de dispositifs tels

que des ordinateurs et des tablettes ne sont pas couverts. Un autre exemple, est l'interface *Qtuio*[1], qui est une interface Qt[2] (cf. s. 4.3.1.1) pour interfaces multi-touches tangibles. Elle fournit une librairie de développement qui redirige des évènements utilisateurs multi-touches et tangibles aux applications de haut niveau, mais il ne permet aucune fonctionnalité de mise en réseau.

Ceci nous a amenés à dire qu'il y a un réel besoin de développer des frameworks d'applications pour environnements multi-dispositifs, incluant des interfaces tactiles multi-touches et tangibles. En ce sens, le framework d'application *Tactileo* serait une réelle contribution pour le développement d'applications, en procurant une solution pour l'unification d'évènements utilisateurs multi-touches et tangibles, et la gestion de la découverte automatique de l'interconnectivité de multiples dispositifs interactifs.

4.3 LE FRAMEWORK TACTILEO

4.3.1 *Architecture et exigences techniques*

Le framework *Tactileo* a pour objectif de fournir une interface de communication logicielle entre dispositifs hétérogènes multiples et applications Qt de haut niveau. Il est conçu sur la base de deux modules indépendants (Figure 26) :

— *Gestionnaire d'événements multi-touches et tangibles* : consiste en un composant unifié pour la gestion d'évènements utilisateurs multi-touches et tangibles. Ce composant repose entièrement sur Qt et utilise le protocole Tangible User Interface Objects (TUIO), qui est lui même fondé sur le protocole Open Sound Control (OSC).

— *Gestionnaire de découverte réseau* : se charge de gérer la connectivité entre plusieurs dispositifs à travers la découverte automatique du réseau. Ce composant est entièrement fondé sur Qt et utilise le protocole Universal Plug and Play (UPnP).

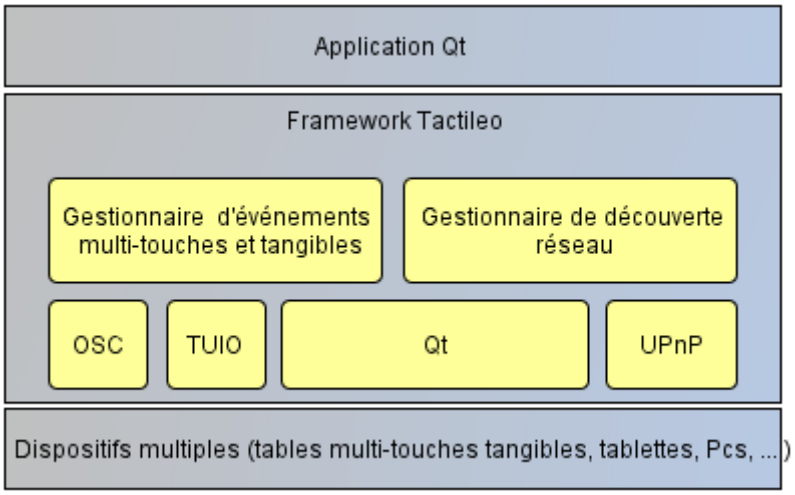


FIGURE 26 – Architecture du framework *Tactileo*.

1. http://qtuio.sirbabyface.net/
2. Qt est une API de développement en C++ : http://www.qt.io/

4.3.1.1 *Utilisation de Qt*

Qt est un framework de développement multiplateformes d'applications et d'interfaces utilisateurs. Il permet la portabilité des applications et supporte plusieurs environnements tels que Windows, Android, iOS, MacOS et Linux. Il fournit une API rendant le développement d'applications pour des dispositifs multiples plus facile et plus rapide [3]. Cela va dans le sens de notre objectif primaire, qui est de fournir des interfaces de programmation de haut niveau pour le développement d'applications multiplateformes.

Qt est également compatible avec le protocole *TUIO*, utilisé dans la gestion d'interaction avec des interfaces utilisateurs multi-touches tangibles. En effet, Qt supporte les dispositifs multi-touches mais pas les interfaces tangibles.

Un autre avantage de Qt est sa distribution sous licence Lesser General Public License (LGPL), qui autorise le développement de logiciels commerciaux. On a choisi de développer avec Qt sous licence LGPL, afin d'avoir le moins de contraintes possibles au regard d'une future exploitation du framework.

4.3.1.2 *Utilisation de TUIO*

TUIO est un framework ouvert qui définit un protocole unifié et une API pour surfaces multi-touches, permettant la transmission d'une description abstraite d'évènements tactiles et d'états d'objets tangibles (Kaltenbrunner et al., 2005).

Le protocole TUIO encode des *données de contrôle* [4] depuis une application de suivi (Tracker), qui est fondée sur un moteur de vision par ordinateur [5] robuste et rapide. Il définit un ensemble de messages redondants qui transmettent de façon constante l'état de la surface interactive aux applications clientes (recevant et exploitant les données transmises) (Figure 27). Ces messages sont communiqués de façon rapide et fiable à l'aide du protocole User Datagram Protocol (UDP).

TUIO fournit des fonctionnalités requises par le framework *Tactileo*, permettant le développement rapide du module de gestion d'événements utilisateurs multi-touches et tangibles.

4.3.1.3 *Utilisation d'OSC*

Le protocole TUIO est encodé suivant le format OSC, qui est un protocole de haut niveau fournissant une méthode efficace d'encodage binaire pour la transmission de données de contrôle (Wright, Freed et Momeni, 2003).

Par défaut, la méthode de transport utilisée par TUIO est l'encapsulation de paquets binaires OSC à travers le protocole UDP. De ce fait, l'implémentation du protocole TUIO implique l'utilisation d'une librairie OSC appropriée, afin de décrypter les messages TUIO et d'écouter les évènements TUIO (Kaltenbrunner et al., 2005). L'utilisation de la librairie OSC est donc primordiale pour le décryptage des messages TUIO.

3. http://www.qt.io/developers/

4. Les données de contrôle sont des données utilisées pour l'identification, la sélection, l'exécution ou la modification d'autre données (Lapedes, 2002)

5. La vision par ordinateur permet à un ordinateur d'analyser et d'interpréter une ou plusieurs images numériques.

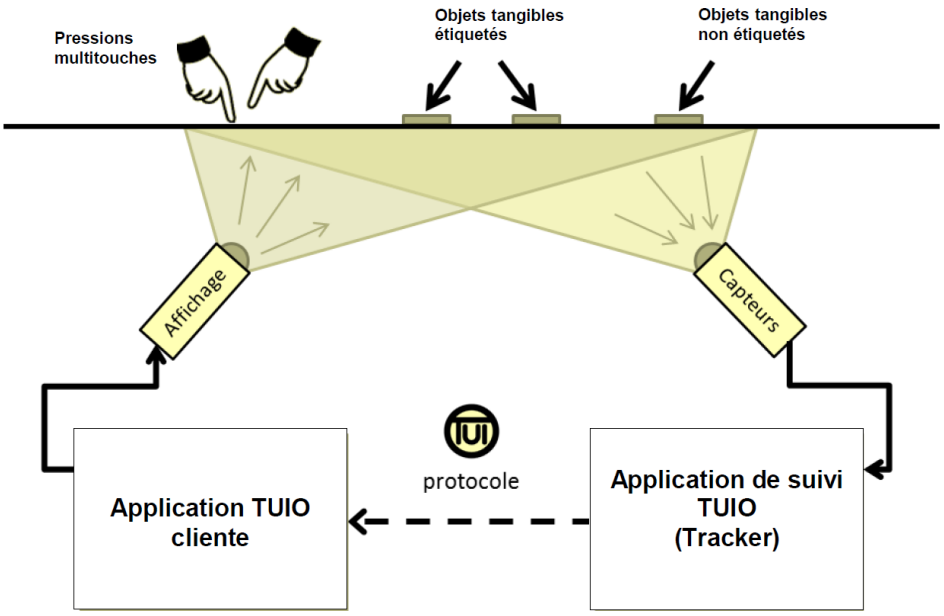


FIGURE 27 – Principe du protocole TUIO.

4.3.1.4 *Utilisation d'UPnP*

La technologie UPnP [6] définit une architecture pour l'interconnectivité en réseau pervasif pair à pair (peer-to-peer ou P2P) d'appareils intelligents, de dispositifs sans fils et d'ordinateurs personnels. Il est conçu de sorte à simplifier la mise en réseau, l'interconnectivité et la découverte automatique d'un grand nombre d'équipements ou d'applications hétérogènes, sans que l'utilisateur se charge de leur configuration. À l'aide de ce protocole, un dispositif peut se joindre de façon dynamique à un réseau, obtenir une adresse IP, communiquer ses capacités et s'informer de la présence et des capacités d'autres dispositifs. Un dispositif peut également quitter facilement un réseau de façon automatique.

Le protocole UPnP est indépendant de tout média, et les dispositifs peuvent être implémentés en utilisant n'importe quel langage de programmation sur n'importe quel système d'exploitation. En ce sens, UPnP est approprié pour le développement du module de gestion de découverte automatique du réseau du framework *Tactileo*.

4.3.2 *Implémentation des protocoles TUIO et UPnP*

4.3.2.1 *Implémentation du protocole TUIO 1.1*

Le gestionnaire d'événements multi-touches et tangibles du framework *Tactileo* implémente la spécification 1.1 du protocole TUIO [7].

Initialement, la spécification de TUIO était centrée sur des objets tangibles étiquetés et sur le suivi de pressions des doigts dans le contexte de surfaces interactives *basées-tables* (KALTENBRUNNER, 2009). Ensuite, ce protocole a connu plusieurs extensions. TUIO

6. http://openconnectivity.org/upnp
7. http ://www.tuio.org/

1.1 est une spécification intermédiaire, incluant des fonctionnalités pour la description d'objets non étiquetés et la possibilité de multiplexer des sources multiples de suivi. La spécification TUIO 1.1 est stable, et a été adoptée par plusieurs projets (Kaltenbrunner, 2009). Elle définit essentiellement deux types d'informations :

— *messages TUIO* : la spécification 1.1 du protocole TUIO encode les événements multi-touches et les événements tangibles dans des messages OSC, et définit deux principaux types de messages : *des messages `SET` et des messages `ALIVE`* (Kaltenbrunner et al., 2005). Les messages `SET` sont utilisés pour communiquer des informations liées à l'état d'un objet, telles que sa position et son orientation. Les messages `ALIVE` indiquent l'ensemble d'objets courants présents sur la surface interactive, en utilisant une liste d'identifiants uniques de sessions.
La spécification TUIO 1.1 définit d'autres messages `FSEQ`, servant à étiqueter de façon unique chaque étape actualisée avec un identifiant unique de trames. Un message optionnel `SOURCE` identifie la source de suivi TUIO, afin de permettre le multiplexage de plusieurs sources de suivi.
Dans notre cas, seuls les messages `SET` et les messages `ALIVE` sont utilisés. Nous ne considérons pas les messages `SOURCE` et les messages `FSEQ`.

— *profiles de messages* : TUIO définit un ensemble de profiles qui permettent la transmission d'un ensemble d'objets courants présents sur la surface interactive, à savoir *cursor, object et blob*, qui désignent respectivement, une pression du doigt (un contact tactile), un objet tangible étiqueté, et un objet générique non étiqueté. Une implémentation OSC encode les messages TUIO comme suit :

```
/tuio/[profileName] set sessionID [parameterList]
/tuio/[profileName] alive [List of active sessionIDs]
/tuio/[profileName] fseq int32
```

À noter que `profileName` (désignation du profile) possède la valeur `2Dobj, 2Dcur`, ou `2Dblb` se référant respectivement à *cursor, object*, ou *blob* dans le contexte de surfaces interactives bidimensionnelles. `sessionID` est l'identifiant unique d'un profile (*cursor, object*, ou *blob*) maintenu durant une session. `parameterList` est la liste d'attributs désignant les propriétés d'un objet telles que sa position et son orientation sur la surface interactive.

4.3.2.2 *Implémentation du protocole de découverte UPnP 1.1*

Le gestionnaire de découverte réseau du framework *Tactileo* implémente le protocole de découverte UPnP défini dans la spécification 1.1 (UPnP 1.1 device architecture) (Presser et al., 2008). La spécification UPnP 1.1 véhicule plusieurs protocoles et requiert plusieurs étapes. Dans notre cas, nous nous basons uniquement sur l'étape de mise en réseau du protocole, à savoir l'étape de *découverte*.

La spécification UPnP 1.1 définit deux grandes catégories de dispositifs : des *dispositifs contrôlés* (controlled devices), appelés simplement *dispositifs* (devices), et des *points de contrôle* (control points) (Presser et al., 2008). Un dispositif possède le rôle de serveur qui est de répondre à toutes les requêtes des points de contrôle. Chacun des dispositifs et point de contrôle peut être implémenté dans plusieurs environnements, incluant

des ordinateurs personnels ou des systèmes embarqués. Plusieurs dispositifs, points de contrôle ou les deux réunis peuvent être implémentés simultanément sur le même terminal réseau.

Avec le protocole de découverte UPnP, quand un dispositif est ajouté au réseau, il annonce ses services à l'ensemble des points de contrôle. De la même manière, quand un point de contrôle rejoint le réseau, il cherche des dispositifs disponibles sur le réseau. L'échange fondamental dans les deux cas, est un message de découverte contenant certaines spécifications concernant le dispositif et ses services. Dans notre cas, nous considérons uniquement, des dispositifs logiques qui ne sont pas intégrés à d'autres dispositifs logiques. Ils sont désignés par des *dispositifs racines* (root devices) dans la spécification 1.1 du protocole UPnP. Nous ne considérons pas non plus les services des dispositifs, notre intérêt porte uniquement sur la présence des dispositifs.

FORMAT DE MESSAGES. Le protocole de découverte UPnP repose sur le protocole Simple Service Discovery Protocol (SSDP) dans l'encodage des messages de découverte, qui utilise à son tour UDP comme protocole de transport. Les dispositifs sont découverts au moyen d'un adressage par diffusion (multicast) sur une adresse IP spécifique (`239.255.255.250`) et sur le port UDP `19 000`. Un message SSDP doit avoir :

— une ligne-début qui doit être spécifiées comme suit : `NOTIFY * HTTP/1.1`, `M-SEARCH * HTTP/1.1`, ou `HTTP/1.1 200 OK`.

— un champ d'entête qui peut être `HOST: 239.255.255.250:1900`.

À noter que les messages `NOTIFY` sont des *annonces* (advertisements), les messages `M-SEARCH` sont des *requêtes de recherche* (search requests), et les messages commençant par `HTTP/1.1 200 OK` sont des *réponses aux requêtes de recherche* (search responses).

ANNONCES. Pour annoncer ses services, un dispositif diffuse un certain nombre de messages de découverte sur le port et l'adresse réservée "`239.255.255.250: 1900`" sur laquelle les points de contrôle écoutent. Étant donné que nous nous intéressons uniquement à la disponibilité des dispositifs sur le réseau, les services ne sont pas considérés. Nous ne considérons également que les messages de diffusion correspondant aux dispositifs racines, les dispositifs embarqués ne sont pas traités. La Figure 28 donne un aperçu de l'implémentation du protocole de découverte UPnP dans le framework *Tactileo*.

Afin de notifier sa disponibilité ou son indisponibilité sur le réseau, un dispositif racine doit diffuser un message comme suit :

```
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age = seconds until advertisement expires
LOCATION: URL to the UPnP description of the root device
NT: notification type
NTS: notification sub-type
SERVER: OS/version UPnP/1.1 product/version
USN: BOOTID.UPNP.ORG: number increased each time a device sends a message
CONFIGID.UPNP.ORG: number used for caching description information
SEARCHPORT.UPNP.ORG: number that identifies port on which a device responds to
    unicast M-SEARCH
```

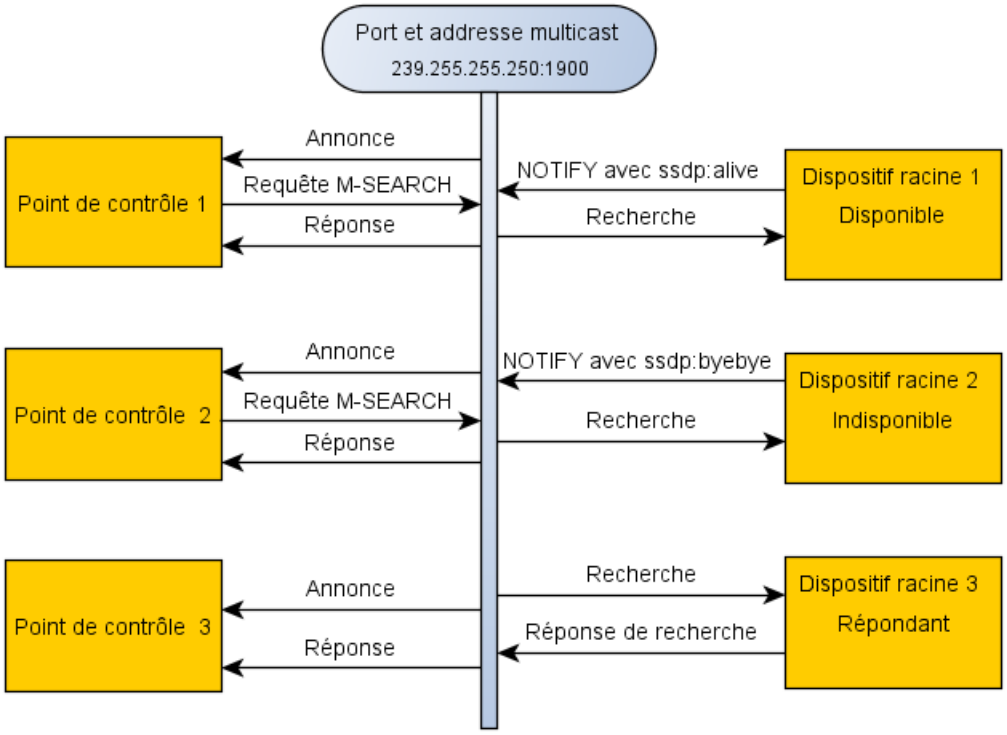


FIGURE 28 – Implémentation du protocole de découverte UPnP 1.1 dans le gestionnaire du réseau du framework *Tactileo*.

À noter que le sous-type de la notification (`NTS`) doit avoir pour valeur "*ssdp :alive*", dans le cas où le dispositif s'ajoute au réseau, ou "*ssdp :byebye*", dans le cas où le dispositif quitte le réseau (Figure 28).

REQUÊTES DE RECHERCHE ET RÉPONSES AUX REQUÊTES DE RECHERCHE. Une façon pour un point de contrôle de rechercher des dispositifs réseau, est d'envoyer une requête multicast avec la méthode `M-SEARCH`, sur le port et l'adresse `HOST: 239.255.255.250:1900` (Figure 28). La requête de recherche doit être formatée comme suit :

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
MAN: "ssdp:discover"
MX: seconds to delay response
ST: search target
USER-AGENT: OS/version UPnP/1.1 product/version
```

À noter que `MAN` définit l'extension de la portée de `HTTP` qui doit être "`ssdp:discover`", et `ST` spécifie la cible recherchée qui doit avoir pour valeur "`upnp:rootdevice`", dans le cas où la recherche porte sur des dispositifs racines. Enfin, le champ `USER-AGENT` spécifie le système d'exploitation et la version du protocole UPnP, et identifie la désignation et la version du produit.

Les dispositifs doivent envoyer des réponses à l'ensemble des points de contrôle qui ont envoyé des requêtes de recherche sur le réseau. Les réponses aux messages `M-SEARCH`

sont parallèles aux annonces, et suivent le même modèle que les messages `NOTIFY` avec "`ssdp:alive`", à l'exception que le champ `NT` est remplacé par le champ `ST`, dont la valeur dépend de celle du champ `ST` de la requête de recherche. La réponse à une requête de recherche est formatée comme suit :

```
HTTP/1.1 200 OK
CACHE-CONTROL: max-age = seconds until advertisement expires
DATE: when response was generated
EXT:
LOCATION: URL for UPnP description for root device
SERVER: OS/version UPnP/1.1 product/version
ST: search target
USN: composite identifier for the advertisement BOOTID.UPNP.ORG: number increased
    each time device sends an initial announce or an update message
CONFIGID.UPNP.ORG: number used for caching description information
SEARCHPORT.UPNP.ORG: number identifies port on which device responds to unicast M-
    SEARCH
```

Dans notre cas, le champ `ST` spécifié dans la requête de recherche porte la valeur "`upnp:rootdevice`", étant donné que nous nous basons uniquement sur des dispositifs racines. De ce fait, un dispositif racine répond une seule fois avec la même valeur du champs `ST` "`upnp:rootdevice`".

4.3.3 *Détails de conception et d'implémentation*

Le framework Tactileo possède une implémentation flexible qui fournit des interfaces de haut niveau pouvant être manipulées en langage C++ en utilisant Qt. Il fournit une librairie OO et a pour objectif de simplifier son utilisation à un programmeur novice. Il est aussi extensible et permet de développer de nouvelles fonctionnalités, et de modifier des fonctionnalités existantes.

4.3.3.1 *Unification d'évènements multi-touches et d'évènements tangibles*

L'unification d'évènements multi-touches et d'évènements tangibles est à la charge du module *gestionnaire d'événements multi-touches et tangibles* (cf. s. 4.3.1). Ce composant du framework possède une architecture modulaire et extensible. Il est conçu sur la base de trois couches indépendantes qui opèrent des évènements utilisateurs multi-touches et tangibles à partir de leur réception jusqu'à leur livraison et propagation (Figure 29) : *couche d'écoute d'événements TUIO, couche de gestion d'évènements, et couche de propagation d'événements*.

LA COUCHE D'ÉCOUTE D'ÉVÉNEMENTS TUIO. Cette couche se charge d'écouter les événements TUIO à chaque fois qu'un paquet UDP est envoyé sur le port 3333 [8]. À leur réception, les messages TUIO sont traités en utilisant les fonctionnalités offertes par la classe `QUdpSocket` fournie par Qt. Les paquets UDP qui encapsulent les messages TUIO

8. Le port 3333 est le port par défaut utilisé par TUIO dans la transmission de paquets UDP contenant les messages OSC.

sont décryptés en utilisant la librairie OSC, tel que requis par le protocole TUIO. Cette couche traite les messages TUIO reçus afin d'identifier les évènements sous-jacents. Cela comprend la vérification des classes des messages (`SET, ALIVE, FSEQ`) et leurs profiles (`2Dobj, 2Dcur, 2Dblb`).

À chaque réception, si le message TUIO est de classe `SET`, alors la couche d'écoute d'événements TUIO crée ou actualise les paramètres d'objets TUIO (`Object, Cursor ou Blob`) en fonction de la valeur du profile du message reçu (`2Dobj, 2Dcur ou 2Dblb`). La couche d'écoute d'événements TUIO constitue et actualise la liste d'objets courants en analysant les messages `ALIVE`. Les objets qui ont été préalablement identifiés et qui ne font plus partie des messages `ALIVE` sont immédiatement retirés de la liste d'objets courants. Tout cela est réalisé pendant que la *couche de gestion d'évènements* est sollicitée pour effectuer la sauvegarde et la mise à jour des événements TUIO, correspondant aux objets TUIO qui sont en cours d'identification, d'actualisation ou d'élimination.

LA COUCHE DE GESTION D'ÉVÈNEMENTS. Une fois les messages TUIO identifiés, la couche de gestion d'évènements s'occupe de la sauvegarde et de la mise à jour des évènements correspondants. Les caractéristiques de chaque classe d'évènements TUIO sont enregistrées et constamment actualisées, en utilisant les données transmises par la *couche d'écoute d'événements TUIO*, qui écoute constamment les changements intervenant sur la surface interactive.

La couche de gestion d'évènements crée une liste d'évènements pour chaque classe d'objets TUIO (`Object, Cursor et Blob`), afin de sauvegarder leur état et leurs paramètres. Ces évènements dérivent de la classe `QInputEvent` fournie par Qt pour la description d'événements. Un évènement désigne l'état d'un objet à un moment donné, et peut avoir l'une des valeurs : `Added, Updated ou Removed` (Ajouté, actualisé ou retiré). De ce fait, tous les évènements sont de type `CURSOR_ADDED, CURSOR_UPDATED, CURSOR_REMOVED, TAG_ADDED, TAG_UPDATED, TAG_REMOVED, BLOB_ADDED, BLOB_UPDATED, ou BLOB_REMOVED`. En plus de leur état, ces évènements possèdent une liste de paramètres (tels que : identifiant, position et orientation) qui sont sauvegardés et constamment actualisés, afin d'être livrés de manière cohérente par la *couche de propagation d'événements*, qui permet l'interfaçage avec des applications de haut-niveau.

LA COUCHE DE PROPAGATION D'ÉVÉNEMENTS. Cette couche ne possède aucun rôle fonctionnel à l'intérieur du framework. Elle est conçue comme une interface externe permettant la communication avec des applications Qt de haut-niveau qui utilisent le framework, dans le traitement d'évènements utilisateurs multi-touches et tangibles. Cette couche est implémentée comme un plugin QT de bas-niveau (un module d'extension ou un greffon), permettant à n'importe quelle application Qt de traiter des évènements multi-touches et tangibles. Ce module offre au travers de quelques interfaces, des fonctionnalités appropriées qui fournissent tous les évènements courants, incluant des informations liées à leurs états et paramètres (cf. s. A.1).

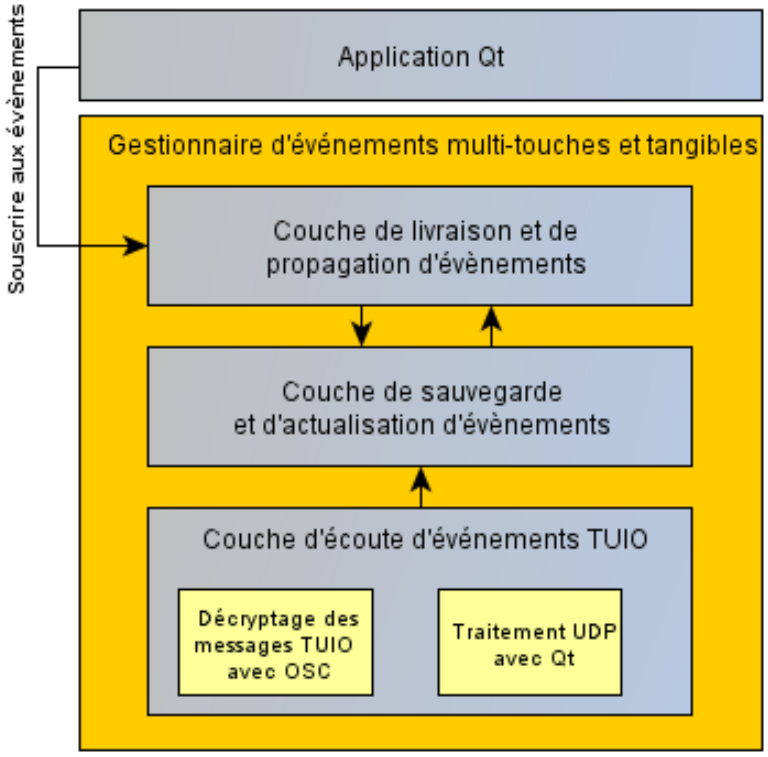


FIGURE 29 – Gestionnaire d'événements multi-touches et tangibles du framework *Tactileo*.

4.3.3.2 *Gestion de découverte automatique du réseau*

Le gestionnaire de découverte automatique du réseau se base sur le protocole de découverte UPnP. Il a pour objectif de retrouver de nouveaux dispositifs venant s'ajouter au réseau, de gérer la modification de leur état de connectivité, et d'informer les autres dispositifs qui sont préalablement découverts des changements intervenus. Ce composant du framework est modulaire et extensible, et complètement développé avec Qt. Il est conçu comme un processus à *threads* [9], et offre une interface externe qui permet aux applications de souscrire au réseau comme des dispositifs, des points de contrôle, ou simultanément des dispositifs et des points de contrôle (Figure 30).

LE PROCESSUS DE DÉCOUVERTE RÉSEAU. Le gestionnaire de découverte réseau est conçu comme un processus à *threads*, dans lequel chaque *thread* possède un rôle spécifique : *contrôle de présence d'interfaces sur le réseau, traitement de messages de découverte, et recherche de dispositifs réseau* (Figure 30).

À chaque fois qu'une application exécutant le framework *Tactileo* sur un terminal réseau donné lance la découverte réseau, le *thread* en charge du *contrôle de présence d'interfaces* est exécuté. Il interroge le réseau toutes les minutes afin d'enregistrer les interfaces nouvellement recensées, et de désinscrire celles préalablement enregistrées mais qui ne sont plus connectées au réseau.

9. Le terme *thread* désigne une unité de traitement dans un processus en informatique.

Dans le cas où une nouvelle interface est trouvée, le *thread* en charge de *traitement des messages de découverte* se met à écouter sur cette interface, et s'apprête à traiter les messages UDP qui transportent les annonces et les requêtes de recherche. Il vérifie l'origine de provenance de chaque message, afin de déterminer si ce dernier provient d'une application exécutant le framework *Tactileo*, et s'assure que les dispositifs racines répondent aux points de contrôle, en envoyant des réponses aux requêtes de recherche.

Dans le cas où le nouveau dispositif découvert agit comme un point de contrôle, le *thread* en charge de la *recherche de dispositifs réseau* est exécuté. Il recherche des dispositifs racines disponibles en envoyant un message `M-SEARCH` en multicast sur le port et l'adresse spécifique (`239.255.255.250:1900`), tel que requis par le protocole de découverte UPnP (Figure 28). Le message de recherche est encodé avec "`upnp:rootdevice`" comme valeur du champs d'entête `ST`, indiquant que la recherche concerne uniquement des dispositifs racines. Le message désigne également "`TactileoFrameworkNetworkCP/1.0`" comme nom et version du produit.

Dans le cas où le nouveau dispositif découvert agit comme un dispositif racine, il diffuse immédiatement une annonce en envoyant des messages `NOTIFY` avec "`ssdp: alive`" comme valeur du champs d'entête `NTS` spécifiant "`TactileoFrameworkNetworkCP`" comme nom du produit. Quand un dispositif racine s'apprête à se déconnecter du réseau, un message `NOTIFY` avec "`ssdp: byebye`" est diffusé à la place (Figure 29). Ces notifications sont diffusées de façon périodique avec un délai de quelque centaines de millisecondes.

Afin de distinguer les dispositifs racines opérant sur le même terminal réseau des dispositifs racines opérant sur un terminal réseau différent, le gestionnaire de découverte réseau définit des *dispositifs locaux* et des *dispositifs distants*. Un terminal réseau inscrit comme dispositif local tout dispositif racine opérant sur ce même terminal, et inscrit comme dispositif distant tout dispositif racine opérant sur un terminal distant. Chaque dispositif réseau est identifié avec un Universally Unique IDentifier (UUID) tel que requis par UPnP.

Le gestionnaire de découverte réseau informe les points de contrôle des changements d'état de connectivité intervenus sur les dispositifs racines, en générant et postant des évènements qui indiquent la présence en réseau des dispositifs distants à un moment donnée. Un évènement peut posséder la valeur `Added ou Removed` (Ajouté ou Retiré). De ce fait, à chaque fois qu'un dispositif racine rejoint le réseau, l'évènement `DEVICE-ADDED` est généré, et à chaque fois qu'un dispositif racine quitte le réseau, l'évènement `DEVICE-REMOVED` est généré à la palce.

L'INTERFACE DE DÉCOUVERTE RÉSEAU. Tout comme la couche de propagation d'événements multi-touches et tangibles, l'interface de découverte réseau est un plugin Qt de bas-niveau, permettant à un dispositif quelconque de réaliser la découverte automatique du réseau. Un dispositif souscripteur peut être une application ou un sous-module d'application. De ce fait, plusieurs composants peuvent souscrire au réseau simultanément au sein d'une même application s'exécutant sur le même terminal réseau. Cette interface offre un certain nombre de fonctionnalités appropriées, permettant à tout dispositif de souscrire au réseau comme étant un dispositif racine, un point

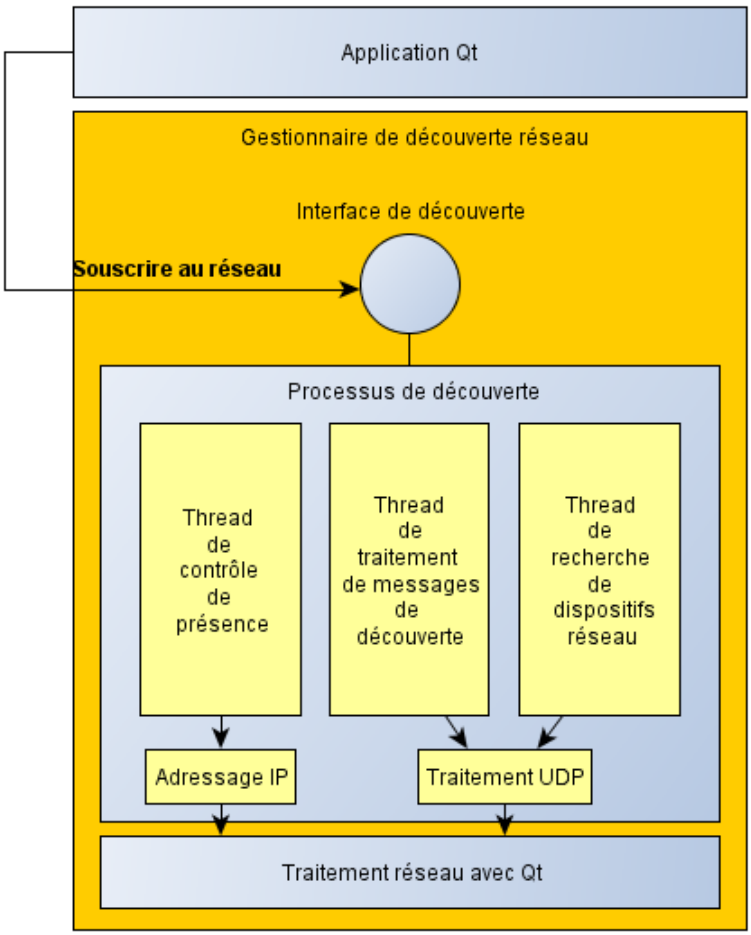


FIGURE 30 – Gestionnaire de découverte automatique du réseau du framework *Tactileo*.

de contrôle ou simultanément un dispositif racine et un point de contrôle, et d'être informé des changements de connectivité intervenus sur d'autres dispositifs (cf. s. A.2).

4.3.4 *Utilisation dans le développement d'applications*

Toute application Qt peut être étendue grâce aux plugins. Cela requiert que l'application détecte et charge les plugins du framework *Tactileo* en utilisant la classe `QPluginLoader` fournie par Qt. Chacun des deux plugins du framework a été compilé séparément en une librairie dynamique, de façon à être détectée et chargée de manière flexible dans une application à l'exécution.

4.3.4.1 *Souscription aux événements multi-touches et tangibles*

Afin d'utiliser le plugin de propagation d'évènements multi-touches et tangibles, une application Qt doit implémenter l'interface du plugin qui fournit plusieurs méthodes permettant de traiter de manière transparente des évènements utilisateurs multi-touches et tangibles, telles que `getAllAliveTags()`, `getAllAliveCursors()` et `getAllAliveBlobs()` (cf. s. A.1). Ces méthodes peuvent être invoquées à l'intérieur

d'une application après souscription aux évènements retournés par le framework, en utilisant la méthode `subscribe()`. Après cela, toutes les propriétés des évènements peuvent être récupérées à l'aide de méthodes appropriées telles que `getTagValue()`, `getCursorId()`, `getBlobArea()` et `getBlobSize()` pour traiter des objets spécifiques. D'autres méthodes communes à l'ensemble des objets TUIO sont également fournies, permettant de récupérer des informations relatives à l'état d'un objet, telles que `getState()`, `getPosition()` et `getOrientation()`. Enfin, une application peut notifier qu'elle a fini d'utiliser le framework, en faisant appel à la méthode `unsubscribe()`.

4.3.4.2 *Souscription au réseau*

Afin d'utiliser le plugin de découverte réseau, une application Qt doit implémenter l'interface du plugin, de la même manière que pour le plugin de propagation d'évènements multi-touches et tangibles. L'interface du plugin de découverte réseau définit plusieurs méthodes, qui permettent à une application de souscrire au réseau pour l'annonce ou la découverte de présence de dispositifs (cf. s. A.2).

À l'aide du protocole UPnP, un dispositif peut souscrire au réseau comme étant un dispositif racine, un point de contrôle ou simultanément un dispositif racine et un point de contrôle. De ce fait, une application peut souscrire au réseau pour lancer la découverte en spécifiant quel type de dispositif UPnP elle opère, en utilisant les méthodes `subscribeAsControlPoint()`, `subscribeAsDevice()`, ou `subscribeAsControlPointAndDevice()`.

Une application peut obtenir la liste des dispositifs locaux ou distants qui sont disponibles, en utilisant les méthodes `getAllLocalDevices()` et `getAllRemoteDevices()`, ou `getLocalDeviceByUuid()` et `getRemoteDeviceByUuid()`. Les caractéristiques des dispositifs peuvent être récupérées à l'aide de méthodes telles que `getName()`, `getUUID()` et `getIPV4Adress()` pour obtenir respectivement le nom du dispositif, son UUID ou son adresse IP. Un dispositif opérant comme un point de contrôle, peut contrôler le changement d'état de présence d'autres dispositifs à un moment donné en écoutant les évènements `DEVICE-ADDED` et `DEVICE-REMOVED`. Enfin, une application peut notifier qu'elle a fini d'utiliser le framework, en faisant appel à la méthode `unsubscribe()`.

4.4 FORCES ET LIMITES DU FRAMEWORK *tactileo*

Le framework *Tactileo* peut être utilisé dans plusieurs domaines d'applications, bien qu'il ait été conçu pour aborder essentiellement les besoins d'applications destinées à être utilisées dans des classes multi-dispositifs. Il est conçu sous forme d'une librairie OO extensible et réutilisable, et facile à utiliser pour le développement de nouvelles applications Qt multiplateformes.

Le framework offre deux interfaces indépendantes. La première fournit des fonctions faciles à utiliser pour la gestion d'évènements utilisateurs multi-touches et tangibles. La seconde permet aux applications s'exécutant sur diverses interfaces interactives d'être facilement interconnectées et de communiquer via un réseau local.

Le framework *Tactileo* ne permet pas la reconnaissance de gestuelles tactiles. D'importantes améliorations peuvent être apportées pour traiter l'interprétation de gestes

tactiles. Un autre aspect qui peut être également amélioré est lié aux fonctionnalités de mise en réseau. Dans l'implémentation actuelle, les communications se font en multicast, à titre d'exemple, les réponses aux requêtes de recherche se font en diffusion multicast, ce qui peut ralentir considérablement la communication réseau avec un nombre important de dispositifs connectés. Cela devrait se faire, dans l'idéal, en unicast en adressant uniquement les points de contrôle qui lancent la recherche, ceux n'ayant pas soumis de requêtes de recherche ne doivent pas être concernés par les réponses. Une autre amélioration peut également concerner la découverte de services, afin d'offrir aux applications d'avantages de capacités de mise en réseau.

4.5 CONCLUSIONS ET RÉSUMÉ DES CONTRIBUTIONS

Le travail décrit dans ce chapitre contribue à l'état de l'art des frameworks d'applications dédiées aux environnements multi-dispositifs, incluant des tables multi-touches tangibles.

Le framework Tactileo offre une interface transparente pour le développement rapide d'applications Qt de haut niveau, dans la gestion d'événements utilisateurs multi-touches tangibles et la gestion automatique du réseau.

Il permet le développement d'applications dédiées à des classes multi-dispositifs telle que la classe Tactileo. Il fournit une librairie de développement et constitue une réelle aide pour répondre à des besoins majeurs de la classe Tactileo, telle que l'interconnexion d'interfaces tactiles et d'autres dispositifs informatiques en synchronisant des ressources pédagogiques de manière transparente.

Cela peut aider à développer des interfaces utilisateurs capables de communiquer entre elles tout en s'exécutant sur des dispositifs hétérogènes (tables multi-touches tangibles, tablettes et ordinateurs) reliés au même réseau local. Cela peut inclure des échanges de données entre un poste enseignant et des postes étudiants/élèves afin de scénariser ou de superviser des activités d'apprentissage. Ces échanges peuvent également opérer entre postes étudiants/élèves afin de permettre la mise en place d'activités collaboratives entre étudiants.

5

CONCEPTION D'UN NOUVEAU MICROMONDE DE PROGRAMMATION ORIENTÉE-OBJET : PROGO

5.1 INTRODUCTION

PrOgO est un micromonde de programmation fondé sur une métaphore de jeu de construction et d'animation 3D pour l'apprentissage de la POO. Il est conçu de façon à aider les étudiants débutants à comprendre les concepts fondamentaux de la POO.

Ce chapitre a pour objectif de décrire l'environnement PrOgO (Section 5.2). Il s'agit de montrer ses fondements conceptuels, c'est-à-dire : 1) la métaphore de construction et d'animation utilisée pour la représentation de concepts fondamentaux de POO et de systèmes OO, 2) les visualisations graphiques illustrant cette métaphore au sein d'un système de représentation transitionnel, 3) et la situation de jeu-play que PrOgO vise à mettre en place. L'objectif étant de montrer comment les concepts inhérents aux micromondes de programmation sont implémentés dans PrOgO. La description de PrOgO comprend son interface utilisateur, et la façon dont est implémentée l'approche didactique par emboîtement hiérarchique des concepts fondamentaux de l'OO, pour leur introduction aux débutants. Bien qu'il soit difficile de montrer tout l'intérêt de PrOgO à l'aide d'une présentation statique de son interface, cette description permet de montrer à quoi ressemble l'interaction avec PrOgO pour un utilisateur final, et comment les notions fondamentales de la POO s'apprennent avec PrOgO. Nous présenterons brièvement comment l'interface de PrOgO peut être contrôlée à distance grâce à un utilitaire indépendant permettant la scénarisation d'activités d'apprenants, et comment un enseignant/chercheur peut activer la fonctionnalité de collecte de traces d'interaction à des fins expérimentales. Nous présenterons un bref historique de son développement et son architecture générale (Section 5.3), avant de donner les détails de sa conception et son implémentation (Section 5.4) reposant sur les technologies Qt et Object-Oriented Graphics Rendering Engine (OGRE) dans le rendu 3D, ainsi que le framework Tactileo dans la gestion d'évènements tactiles et la découverte automatique du réseau. Nous présenterons son système de visualisation et de rendu 3D, les techniques d'interaction 3D implémentées, la façon dont il procède dans l'analyse de code C++ et la façon dont il implémente les interfaces de propagation d'événements utilisateurs multi-touches et de découverte automatique du réseau du framework Tactileo. Nous soulignerons quelques unes de ses forces et limites (Section 5.5), et nous conclurons avec nos principales contributions (Section 5.6).

5.2 DESCRIPTION DE L'ENVIRONNEMENT PROGO

5.2.1 *Métaphore, système de représentation transitionnel et jeu-play*

MÉTAPHORE DE LA CONSTRUCTION ET D'ANIMATION. L'un des objectifs conceptuels de PrOgO consiste à trouver une métaphore qui soit :

— *fidèle* dans la description des concepts de POO et suffisamment *puissante* pour réduire la distance avec les concepts du monde réel,

— *consistante*, c'est-à-dire, capable de décrire non seulement des concepts de base liés à l'objet et à la classe, mais aussi d'autres concepts fondés sur ces deux derniers, et donc qui permette de décrire non seulement des concepts de POO élémentaires, mais aussi des systèmes OO complexes décrits par des classes et des relations d'interaction entre classes.

Afin d'arriver à une métaphore qui codifie de manière précise et fidèle les concepts de POO, PrOgO s'inspire des micromondes de POO existants, dans lesquels le concept d'objet est expérimenté comme un *objet actif* dans un monde graphique restreint, et le concept de la classe comme une *description des propriétés et des comportements de l'objet*. Les objets informatiques sont représentés de façon métaphorique comme des objets *physiques et de société*. Par analogie aux objets physiques, les objets informatiques possèdent une apparence et un comportement, et par analogie aux objets de société, ils possèdent la capacité de communiquer entre eux et de répondre aux actions de l'utilisateur (cf. s. 3.4.4.1). A la métaphore *d'objet actif* est associée *la métaphore d'animation* qui permet de penser en termes d'actions, d'objectifs et de fonctions. Ce qui constitue des principes fondamentaux de la programmation.

Afin d'arriver à une métaphore consistante qui permet de décrire des systèmes OO complexes, PrOgO repose sur *la métaphore de la construction*. Par analogie à un composant structurel tel qu'un composant de jeu de construction, un objet informatique peut se lier à un autre objet afin de constituer une structure plus complexe ayant suffisamment de connaissances et de compétences pour agir dans un environnement donné. Ce choix conceptuel part du principe que chaque objet informatique consiste en une entité active dans un programme OO, et que l'objectif fonctionnel de ce dernier est réalisable grâce aux attributs et méthodes contenus dans chaque objet, et aux relations d'interaction entre objets. La métaphore de la construction est employée car elle permet de représenter des entités élémentaires et de constituer des systèmes complexes à partir de ces entités élémentaires, et donc d'atteindre l'extensibilité et la consistance dans la représentation et la description des concepts fondamentaux de l'OO .

Ce qui vient appuyer notre choix d'utilisation de la métaphore de la construction comme support de description des concepts de base de la POO, est la description de cette métaphore donnée par ETCHEGOYEN (2004). Dans son article, "*La métaphore de la construction*", il aborde cette dernière d'un point de vue philosophique en montrant son lien avec la formation (ETCHEGOYEN, 2004). Il souligne le sens du verbe *construire* qui décèle par de multiples usages l'idée d'*ériger*, de *structurer* et de *vertébrer*. Il précise que le concept de construction est lié à celui de l'*instruction* et n'en est guère éloigné, puisque, dans la langue latine plus encore que dans la langue française, *instruire signifie édifier*. L'instruction, pédagogique soit-elle, se réfère directement au savoir structurant. Il sou-

ligne également que le concept de construction est très proche de celui d'*organisation*, en prenant pour exemple les jeux de construction tels que *Meccano* et *Lego* qui sont considérés comme *éminemment éducatifs*. Nous retenons essentiellement de cette déclaration ce qui suit :

— la métaphore de la construction est très puissante et possède une dimension éminemment éducative, car elle est par essence liée aux concepts d'*instruction* et d'*organisation*.
— les jeux de construction permettent d'illustrer cette métaphore, au moyen d'entités concrètes, servant à édifier des systèmes organisationnels et à structurer des savoirs.

Cela nous amène à dire que la métaphore de la construction est appropriée pour illustrer des concepts et des systèmes OO, qui sont des systèmes organisationnels structurés et formels.

De son coté, Barbier (2009) explique qu'une entité logicielle est par essence un artefact, un élément artificiel qui imite incomplètement un concept, une chose physique ou logique, une idée, ...etc tous issus d'une "*réalité observée*". Il ajoute : *l'idée-clé des langages à objets a toujours été de fournir un support de description des choses perçues en espérant un fossé plus étroit avec le monde réel*. On peut souligner deux idées essentielles sous-jacente à cette déclaration :

— La première concerne l'objectif majeur de la POO, qui est de décrire des phénomènes du monde réel.
— La seconde concerne les langages de POO, qui tentent de réduire la distance avec des concepts de l'expérience sensible, dans la description des phénomènes du monde réel.

Par ailleurs, ce deuxième objectif est difficile à atteindre. Les langages de POO fournissent des moyens techniques permettant de modéliser des problèmes issus d'une réalité observée. Or, c'est souvent à ce niveau que les apprenants débutants éprouvent des difficultés d'apprentissage. La métaphore de la construction, tente ici de palier cette difficulté en partant d'un exemple d'une réalité observée, qui est une structure composite, afin d'amener l'apprenant débutant à construire sa compréhension des concepts formels de la POO édifiés et organisés au sein de cette structure concrète. Cette structure possède des propriétés communes avec les concepts fondamentaux de la POO. Le rôle de l'apprenant est de construire cette structure à partir de composants concrets. L'ensemble de ses interactions lors de la réalisation de cette construction, lui permettent d'accéder aux propriétés des concepts de programmation formels.

SYSTÈME DE REPRÉSENTATION TRANSITIONNEL. L'une des propriétés fondamentales des micromondes de programmation est de fournir à l'apprenant la possibilité d'interagir avec des objets transitionnels, qui sont des entités visuelles ayant des propriétés communes à celles de concepts de programmation. Ces objets transitionnels font d'un micromonde de programmation un système de représentation transitionnel, car ils permettent d'établir un lien sémantique fort entre des concepts formels de la programmation et des objets concrets du monde réel (ou de l'expérience sensible) (cf. s. 3.5.2.1).

PrOgO est un système de représentation transitionnel, dans lequel les objets transitionnels sont des objets 3D concrets, interactifs et visibles à l'interface, ayant un comportement qui rappelle celui des concepts fondamentaux de la POO. L'idée sous-jacente est de fournir un moyen de réduire la distance entre les objets manipulés à l'interface et les concepts de POO visés, et donc de combler l'écart entre l'apprentissage intuitif et l'apprentissage formel.

Les concepts formels de la POO que la version actuelle de PrOgO permet d'aborder sont des concepts fondamentaux liés à l'*objet* et à la *classe* (Tableau 6).

Concepts liés à l'*objet*	Concepts liés à la *classe*
Lien existant entre l'objet et la classe : un objet est une instance de classe	Rôle d'une classe : une classe sert à décrire un nouveau type de données par la déclaration et la définition de ses données et fonctions membres
Caractéristiques d'objet : un objet se caractérise par des attributs et des méthodes	Encapsulation dans une classe : une classe encapsule ses données et fonctions membres, afin de contrôler leur accès depuis l'extérieur et de préserver leur intégrité
Modification d'état d'objet : modifier la valeur d'un attribut d'objet ou réaliser un appel de méthode sur l'objet, entraîne la modification de l'état de cet objet	Constructeur de classe : un constructeur de classe permet d'initialiser son instance (l'objet) à sa création

Tableau 6 – Concepts formels de la POO visés par PrOgO v.0.7.

PrOgO permet d'accéder à l'ensemble des concepts visés par le biais d'un ensemble d'objets transitionnels qui sont visualisés graphiquement en trois dimensions afin d'assurer leur aspect concret. Chaque graphique élémentaire 3D est une représentation visuelle d'un objet informatique. PrOgO fournit des modèles d'objets (composants graphiques 3D) représentatifs de classes. Les objets s'obtiennent par instanciation de ces modèles 3D. Chaque instance va avoir la même structure visuelle que sa classe et la même apparence à la création. L'utilisateur peut ensuite changer l'apparence de cette instance en accédant à ses attributs et en invoquant ses méthodes. L'apparence d'un objet est définie par sa position (son emplacement par rapport à un autre objet), sa couleur ou sa rotation. Son comportement est défini par deux fonctionnalités, la possibilité de changer de couleur pendant un temps donné et/ou la possibilité de réaliser une rotation pendant une certaine durée. Enfin, chaque objet peut être assemblé à un autre objet afin de construire une structure plus complexe.

Les objets visualisés à l'interface sont à l'image d'instances de classes pouvant être décrites par deux catégories de classes abstraites qui ne sont pas visualisées à l'interface [1], à savoir :

— une classe appelée `ComposantStructurel` servant à décrire l'ensemble des *composants structurels* pouvant être utilisés lors de la réalisation de la construction 3D. Cette classe possède un unique attribut `couleur` et deux méthodes `connecter()` et `colorierPendant()` (Figure 31). Ces deux méthodes permettent respectivement de lier un objet à un autre et de colorier un objet pendant un temps donné.

— une classe appelée `ComposantActif` servant à décrire l'ensemble des composants structurels *actifs*. Ces composants sont dit *actifs* car ils sont capables d'effectuer un mouvement (une action de rotation) pendant un temps donné, et d'entraîner

1. Une classe abstraite n'étant pas instanciable, l'utilisateur ne doit pas la visualiser au même titre que les classes instanciables.

avec eux les composants qui sont directement connectés à eux. Cette classe dérive de la classe `ComposantStructurel` héritant ainsi de ses caractéristiques, et possède un attribut additionnel `angleDeRotation` (*angle de rotation*), et une méthode `tournerPendant()` qui permet de faire tourner un objet d'un angle donné pendant un temps donné (Figure 31).

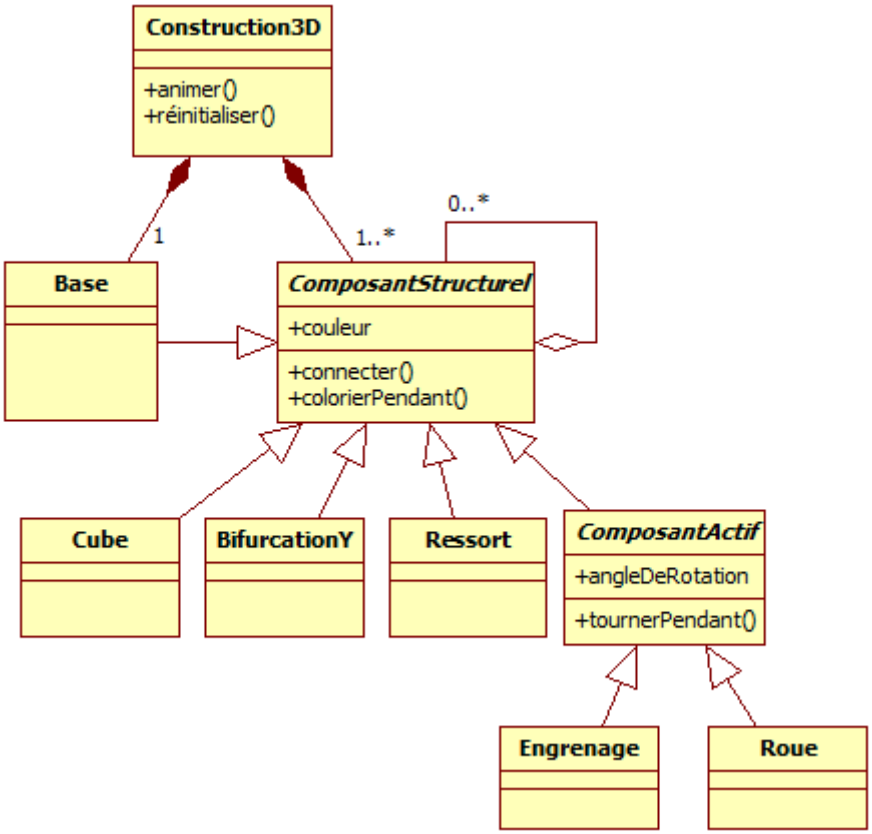
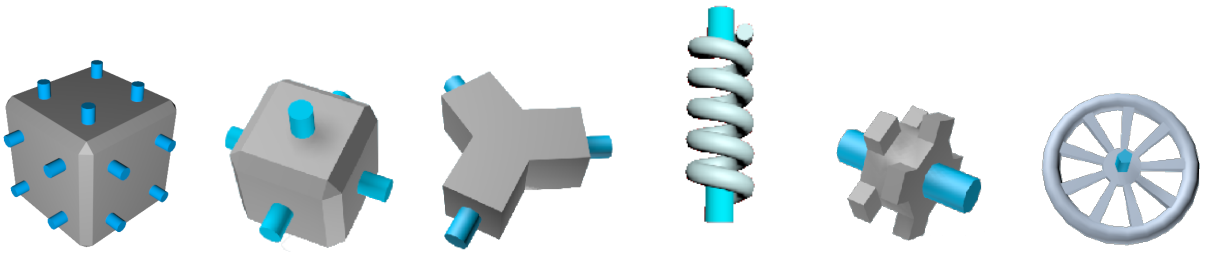


FIGURE 31 – Diagramme de classes UML spécifiant une construction 3D dans PrOgO.

Les classes visualisées à l'interface sont des classes instanciables pouvant donner lieu à des objets manipulables dans l'interface. Ces classes sont de deux types, des classes dérivant de la classe `ComposantStructurel`, à savoir `Base, Cube, BifurcationY, et Ressort`, et des classes dérivant de la classe `ComposantActif`, à savoir `Engrenage et Roue` (Figure 31). À noter que la classe `Base` existe en une seule instance et représente la base de construction, les classes restantes sont instanciables un nombre de fois indéterminé et leurs instances constituent les entités élémentaires de la construction finale, c'est-à-dire, les données membres de la classe représentative de cette dernière.

L'ensemble des composants 3D représentatifs de ces classes et de leurs instances constituent les objets transitionnels que l'utilisateur peut directement manipuler et utiliser pour créer d'autres objets transitionnels (Figure 32).

Une structures 3D construite par l'utilisateur est représentative d'un objet composite, dont les composants comprennent l'instance unique de la classe `Base` (la base de construction), et l'ensemble des composants structurels et/ou actifs que l'utilisateur a employé lors de sa construction. Chaque construction 3D constitue une instance d'une nouvelle classe et consiste en un système OO qui peut être décrit par un diagramme de



(a) Base. (b) Cube. (c) BifurcationY. (d) Ressort. (e) Engrenage. (f) Roue.

FIGURE 32 – Objets transitionnels (composants structurels [a,b,c,d] et composants structurels actifs [e,f]) associés aux notions de *classe* et d'*objet* dans PrOgO.
N.B. À la création, les objets possèdent la même apparence visuelle que leurs classes respectives.

classes Unified Modeling Language (UML) spécifiant les caractéristiques des différentes classes utilisées lors de sa réalisation ainsi que leurs relations d'interaction [2] (Figure 31).

Une construction 3D possède également deux méthodes : la méthode `animer()` qui regroupe l'ensemble des comportements d'animation programmés par l'utilisateur lors de la réalisation de la construction, à savoir les appels des méthodes `colorierPendant()` et `tournerPendant()`. La méthode `réinitialiser()` qui permet de réinitialiser la construction 3D à son état de création suite à des modifications éventuelles effectuées par l'apprenant.

JEU-PLAY. PrOgO est conçu de sorte à permettre la mise en place d'une situation de *jeu-play*, dans laquelle l'apprenant peut acquérir de façon incrémentale les concepts fondamentaux de la POO, et développer des compétences impliquant la capacité de manipuler ces concepts. La métaphore de construction et d'animation est illustrée dans un jeu de construction et d'animation 3D. L'objectif du jeu-play consiste donc à construire et à animer des robots ou des structures mécaniques 3D.

Dans cette situation de jeu-play, PrOgO offre un espace de *créativité* et d'*imagination*, dans lequel l'apprenant est amené à faire appel à sa propre créativité et à son imagination, afin de réaliser des structures robotiques animées. Cela conduit à un apprentissage naturel et intuitif. Il s'agit d'un apprentissage qualifié *d'apprentissage par conception, ou d'apprentissage par construction (ou constructiviste)* (cf. s. 3.4.4.4).

Cette situation de jeu-play découle de la métaphore de la construction et d'animation employée. Elle se met en place quand l'apprenant interagit avec l'environnement PrOgO, qui est un espace de créativité et d'imagination. PrOgO offre un feedback dynamique, permettant à l'apprenant de produire du sens sur les concepts abstraits de la POO, par l'expérience, la découverte, la manipulation et l'observation.

L'interaction avec PrOgO doit également impliquer un sentiment de divertissement et de satisfaction qui se crée chez l'apprenant.

2. À noter que ces relations d'interaction ne sont pas présentées à l'utilisateur dans la version actuelle de PrOgO.

5.2.2 *Interface utilisateur*

L'interface de PrOgO (Figure 33) dispose en son centre d'une scène 3D créative (*cadran 1*) dans laquelle se trouve en permanence le bloc 3D *base*, représentant la base de construction sur laquelle viendront se connecter les autres objets créés par l'utilisateur pour constituer une structure 3D (un robot ou une machine 3D). En bas à gauche de l'interface (*cadran 2*) se trouve l'onglet "*Classes*" qui liste les blocs de construction 3D représentant les classes instanciables. En haut à gauche de l'interface (*cadran 3*) est affichée l'arborescence des objets venant d'être créés et assemblés dans la scène 3D. L'onglet de droite (*cadran 4*) abrite l'éditeur de code à autocomplétion qui contient un fichier nommé `main.cpp` affichant le programme principal, dans lequel est généré le code C++[3] correspondant aux actions directes de l'utilisateur sur les objets dans la scène 3D. Parallèlement, le résultat de la modification de code dans cet éditeur est instantanément retranscrit sur les objets dans la scène 3D. Au lancement de PrOgO, l'éditeur affiche le code correspondant à l'état initial de la scène 3D. La fonction `main()` affiche la déclaration de l'objet `base` et la classe `Base.hpp` est inclue dans l'entête du fichier `main.cpp` comme suit :

```
#include "Base.hpp"

void main () {
        Base base;
}
```

En bas de l'éditeur se trouve deux boutons, le premier sert à la réinitialisation du contenu du programme principal qui entraîne simultanément la réinitialisation de la scène 3D, et le second sert à la création d'une nouvelle classe et le passage vers le second mode. PrOgO dispose également d'une barre de menus (*cadran 5*) qui permet principalement la création et la sauvegarde de nouveaux projets, l'annulation et le rétablissement des actions réalisées à l'intérieur de la scène 3D ou de l'éditeur de code, de masquer/démasquer les connecteurs de construction visualisés sur les blocs 3D, et de lancer l'exécution pas à pas de code et l'exécution de l'animation programmée sur la réalisation 3D.

L'édition de code se fait par autocomplétion aux lignes vertes et un assistant de contenu personnalisé aide l'utilisateur dans la rédaction des instructions. Ce mécanisme a été choisi car il est d'une part, le plus simple à mettre en place tout en permettant de fournir rapidement aux utilisateurs une liste de propositions d'instructions à exécuter. D'autre part, l'autocomplétion de code est mieux adaptée à l'interaction tactile du point de vue de l'utilisateur. Enfin, cela permet de limiter la syntaxe utilisée à celle des concepts OO amenés de façon progressive dans PrOgO et de mettre l'accent sur les concepts et non pas sur le langage de programmation.

L'autocomplétion se déclenche suite à un clic souris, une touche de clavier ou un contact tactile. L'éditeur contient deux zones de code éditables, une zone d'entête réservée à l'inclusion de fichiers (`*.hpp`) et une zone de code source. L'assistant de contenu est constitué de façon dynamique, en fonction du code entré aux lignes éditables.

3. PrOgO génère et analyse du code C++ selon la norme C++11

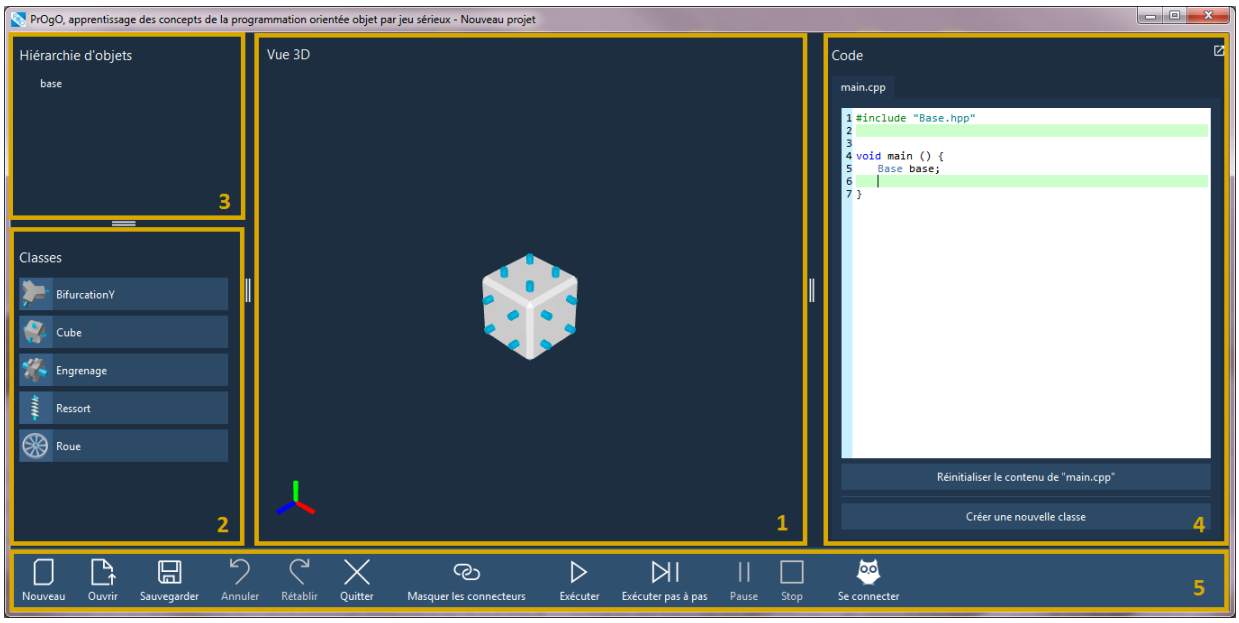


FIGURE 33 – Interface principale de PrOgO.

Dans la scène 3D, il est possible de tourner autour de la construction 3D en déplaçant la souris avec la molette enfoncée (ou par déplacement d'une pression tactile), et de zoomer sur la construction 3D en tournant la molette souris (ou par un pincement tactile). Il est possible de visualiser la trace d'exécution des instructions du programme principal en cliquant successivement sur le bouton *Exécuter pas à pas*. Cela va entraîner simultanément la surbrillance jaune des instructions en cours d'exécution, et la visualisation du résultat de leur exécution dans la scène 3D. La visualisation de l'animation est provoquée en appuyant sur le bouton *Exécuter*.

5.2.3 *Implémentation d'une approche didactique par emboîtement hiérarchique des concepts fondamentaux de la POO*

La manipulation des concepts POO dans PrOgO se fait de façon directe dans la scène 3D ou dans l'éditeur à autocomplétion en utilisant des instructions de code C++. Les concepts sont introduits de façon emboîtée et progressive suivant deux modes successifs. Le premier est consacré à *l'utilisation d'objets* et le second à *la création de classes* (cf. s. 2.6.4).

5.2.3.1 *Utilisation d'objets*

Le premier mode dédié à l'utilisation d'objets se concentre sur les concepts liés à l'objet. L'objectif étant de permettre à l'apprenant de maîtriser l'objet, avant de s'initier à d'autres concepts plus complexes directement fondés sur l'objet. Lors de cette première étape d'apprentissage, les étudiants sont amenés à utiliser des objets afin de découvrir les concepts inhérents à l'objet.

CRÉATION D'OBJETS, INSTANCES DE CLASSES. Afin de créer un objet dans la scène 3D, il suffit de choisir la classe à instancier dans l'onglet "*Classes*", à l'aide d'un simple clic gauche de la souris (ou d'un toucher tactile). Cela va créer un objet qu'il faudra connecter à la base de construction ou à un autre objet préalablement connecté dans la scène, en sélectionnant un de ses connecteurs. Les connecteurs servent d'assistants visuels qui permettent d'assembler facilement les composants graphiques dans la scène 3D. L'objet créé apparaît instantanément dans la hiérarchie d'objets qui s'affiche dans le *cadran 3* de l'interface (Figure 33). Les nœuds parents dans l'arbre d'objets sont des objets *actifs* (les articulations de la construction). Cette action de création graphique entraîne la génération de deux instructions de code correspondant à l'instanciation de la classe choisie et à l'appel de la méthode `connecter()` par l'objet créé. À titre d'exemple, la création d'une instance de la classe `Cube` et son placement sur l'objet `base` entraîne la génération de nouvelles instructions et la modification du programme principal comme suit :

```
#include "Base.hpp"
#include "Cube.hpp";

void main () {
        Base base;
        Cube cube_0;
        cube_0.connecter(0,base,5);
}
```

Un nom par défaut `cub_0` est attribué à l'objet créé que l'utilisateur peut modifier ensuite dans le code. Les connecteurs d'objets possèdent des identifiants numériques `(0, 1, 2, ...)`. L'instruction «`cube_0.connecter(0,base,5);`» signifie que l'objet `cube_0` est rattaché par défaut par son connecteur `0` à l'objet `base`, et l'objet `base` se retrouve ainsi connecté par son connecteur `5` à l'objet `cube_0` (Figure 34).

FIGURE 34 – Création d'objets dans la scène 3D et génération automatique de code C++ correspondant.

La création d'objets dans l'éditeur de code revient à d'abord ajouter l'instruction d'inclusion de la classe à instancier dans l'entête du fichier `main.cpp`. La classe inclue s'ajoute à la liste des propositions de l'assistant de contenu, et l'utilisateur a ensuite la possibilité de l'instancier dans le corps de la fonction `main()` (Figure 35). Le nom de l'objet est entré à l'aide d'une boîte de dialogue. Avant d'être visualisé, le nouvel objet créé doit être d'abord connecté à un autre objet présent dans la scène 3D.

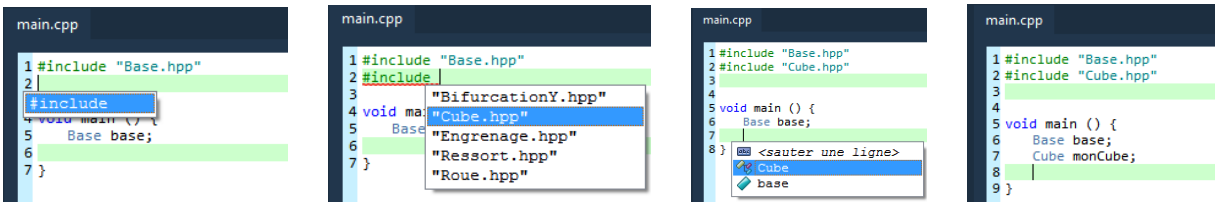


FIGURE 35 – Création d'objets dans l'éditeur de code.

ACCÈS AUX ATTRIBUTS ET MÉTHODES D'OBJETS ET MODIFICATION D'ÉTATS D'OBJETS. Un clic droit ou une pression du doigt maintenue sur un objet dans la scène 3D permet de le sélectionner. Une boîte englobante s'affiche autour de l'objet pour indiquer que l'objet est bien sélectionné. Cela entraîne au même moment l'apparition d'un menu déroulant qui invite à sélectionner les caractéristiques de l'objet, c'est-à-dire ses attributs et ses méthodes. L'entête du menu déroulant affiche également l'identité de l'objet sélectionné, c'est-à-dire son nom et sa classe «`nomObjet:NomClasse`». La sélection d'un attribut d'objet entraîne l'affichage d'un second menu invitant à choisir une nouvelle valeur d'attribut. L'entête de ce sous-menu affiche également la valeur courante de l'attribut sélectionné. De la même manière, la sélection d'une méthode d'objet entraîne l'affichage d'un second menu invitant à définir les paramètres de cette méthode. L'utilisateur peut également observer que les objets ne comportent pas tous les mêmes attributs et les mêmes méthodes (Figure 36).

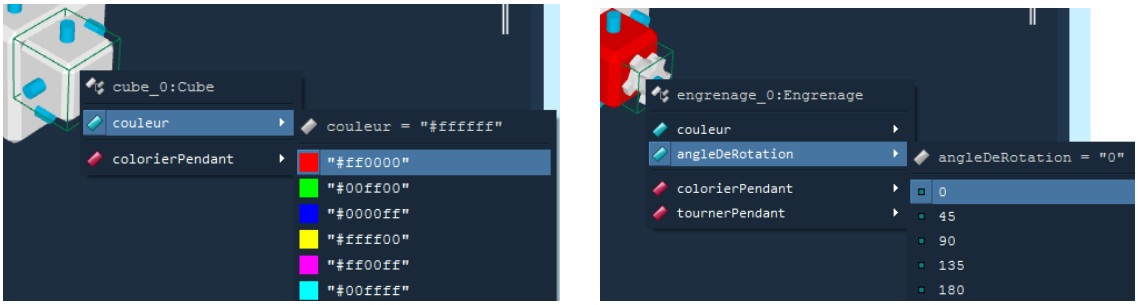


FIGURE 36 – Accès aux attributs et méthodes d'objets dans la scène 3D.

Dans l'éditeur de code, une fois un objet créé, l'assistant de contenu ajoute à sa liste de propositions le nom de ce dernier. Quand l'utilisateur entre le nom de cette variable dans le code, l'assistant de contenu lui propose la liste des attributs et des méthodes de cet objet (Figure 37).

La modification d'une valeur d'attribut ou l'exécution d'un appel de méthode sur l'objet, fait modifier l'état de ce dernier. Cela se fait aussi bien dans la scène 3D (ce qui entraîne la génération de code C++ associé) que dans l'éditeur de code (le résultat d'exécution des instructions est visualisé dans la scène 3D). Le paramétrage des méthodes d'objets se fait en partie par le biais de boîtes de dialogue (Figure 38). Le code suivant montre un exemple de manipulation d'attributs et de méthodes dans PrOgO :

```
#include "Base.hpp"
#include "BifurcationY.hpp"
#include "Engrenage.hpp"
```

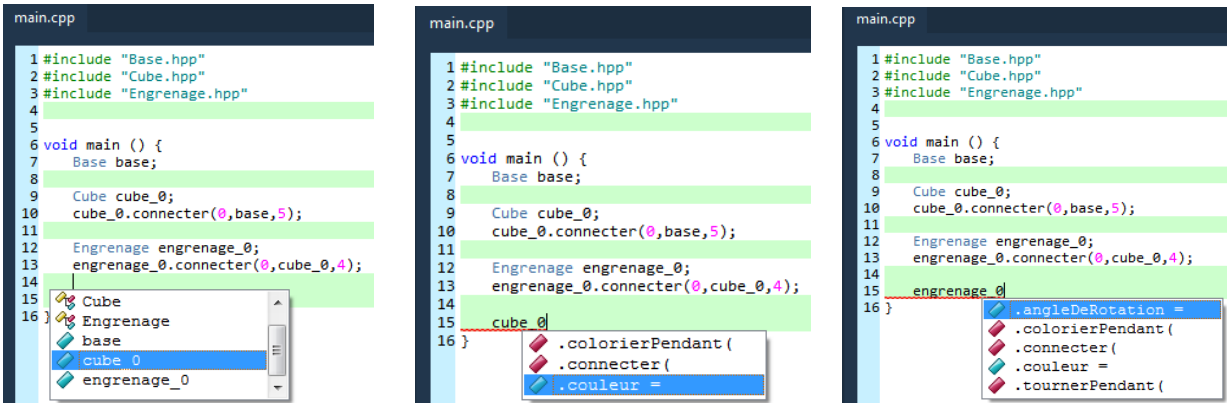


FIGURE 37 – Accès aux attributs et méthodes d'objets dans l'éditeur de code.

```
void main () {
        Base base;
        BifurcationY bifurcationY;
        bifurcationY.connecter(0,base,21);
        Engrenage engrenage;
        engrenage.connecter(0,base,8);
        Engrenage engrenageBis;
        engrenageBis.connecter(0,base,9);

        engrenage.couleur = "#0000ff";
        engrenage.angleDeRotation = 90;
        bifurcationY.colorierPendant("#55aa00",10);
        engrenageBis.tournerPendant(90,4);
}
```

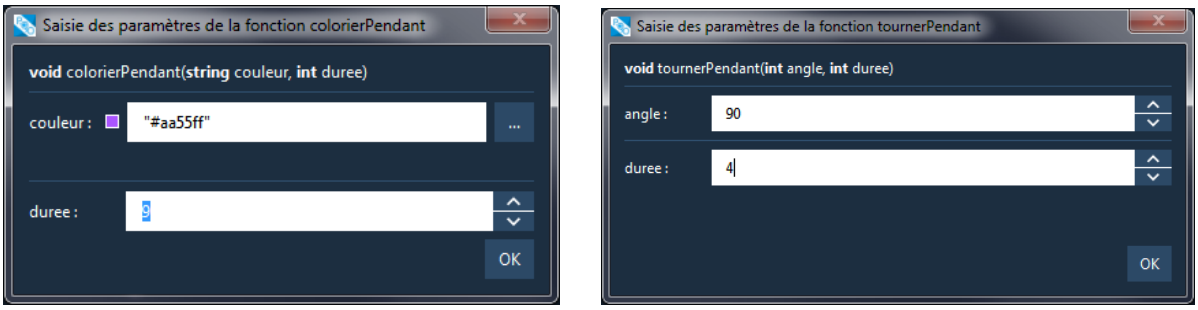


FIGURE 38 – Définitions des paramètres de méthodes par le biais de boîtes de dialogue.

5.2.3.2 *Création de classes*

Le second mode dédié à la création de classes se concentre sur les concepts liés à la classe. Lors de cette étape d'apprentissage, les étudiants sont amenés à créer des classes afin de découvrir les concepts sous-jacents.

ABSTRACTIONS, NOUVEAUX TYPES DE DONNÉES. Toute nouvelle construction dans PrOgO constitue une nouvelle classe que l'utilisateur peut créer en appuyant sur le bou-

ton «*Créer une nouvelle classe*» se trouvant en bas de l'éditeur de code (Figure 33, *cadran 4*). Cela entraîne l'apparition d'une boîte de dialogue invitant à saisir le nom de la classe à créer. Dès qu'une classe est créée, elle est ajoutée à un nouvel onglet «*Mes classes*» qui apparaît sous l'onglet «*Classes*» pour être instanciée dans la scène 3D autant de fois que l'utilisateur le souhaite (cf.s. C.3). Il s'agit d'une nouvelle abstraction de données, ou d'un nouveaux type de données qui existe désormais dans PrOgO que l'apprenant peut manipuler. L'apprenant peut ainsi créer plusieurs classes différentes et obtenir différentes instances qu'il pourra visualiser dans la scène 3D. La création d'une nouvelle classe entraîne également l'apparition d'un nouveau programme principal contenant une fonction `main()` vide, ainsi que deux nouveaux fichiers, l'un contient la déclaration de la nouvelle classe (`*.hpp`) et le second la définition de cette dernière (`*.cpp`) (Figure 39).

main2.cpp | Robot.hpp | Robot.cpp

```hpp
#pragma once

#include "Base.hpp"
#include "BifurcationY.hpp"
#include "Cube.hpp"
#include "Engrenage.hpp"


class Robot{

private:

    Base base;
    BifurcationY bifurcationy_0;
    Cube cube_0;
    Engrenage engrenage_0;

    void initialiserAngles();
    void initialiserCouleurs();

public:
    Robot();

    void animer();
    void reinitialiser(bool b_angle, bool b_color);
};

```

(a) Fichier de déclaration.

main2.cpp | Robot.hpp | Robot.cpp

```cpp
#include "Robot.hpp"

Robot::Robot(){
    bifurcationy_0.connecter(0,base,5);
    cube_0.connecter(0,bifurcationy_0,1);
    engrenage_0.connecter(0,cube_0,4);

    initialiserAngles();
    initialiserCouleurs();
}

void Robot::initialiserAngles(){
    engrenage_0.angleDeRotation = 0;
}

void Robot::initialiserCouleurs(){
    base.couleur = "#ffffff";
    bifurcationy_0.couleur = "#ffffff";
    cube_0.couleur = "#ffffff";
    engrenage_0.couleur = "#0000ff";

}

void Robot::animer(){
    bifurcationy_0.colorierPendant("#5420ff",10);
    engrenage_0.tournerPendant(45,3);

}

void Robot::reinitialiser(bool b_angle, bool b_color){
    if (b_angle)
        initialiserAngles();
    if (b_color)
        initialiserCouleurs();
}
```

(b) Fichier de définition.

FIGURE 39 – Accès aux fichiers de déclaration et de définition d'une nouvelle classe dans PrOgO.

Dans l'éditeur de code, l'utilisation d'une nouvelle classe dans le programme principal se fait de la même manière que dans le mode dédié à l'utilisation d'objets. Il s'agit de créer des instances de cette classe et d'accéder à ses deux méthodes publiques `animer()` et `réinitialiser()` (Figure 40), ses attributs membres étant privés, ils ne sont accessibles que depuis les fichiers de déclaration et de définition de la classe. Le code suivant montre un exemple de manipulation d'une instance d'une nouvelle classe `Robot` dans le programme principal :

```
#include "Robot.hpp"
2
void main () {
        Robot robot;
```

```
        robot.animer();
        robot.reinitialiser(true,true);
}
```

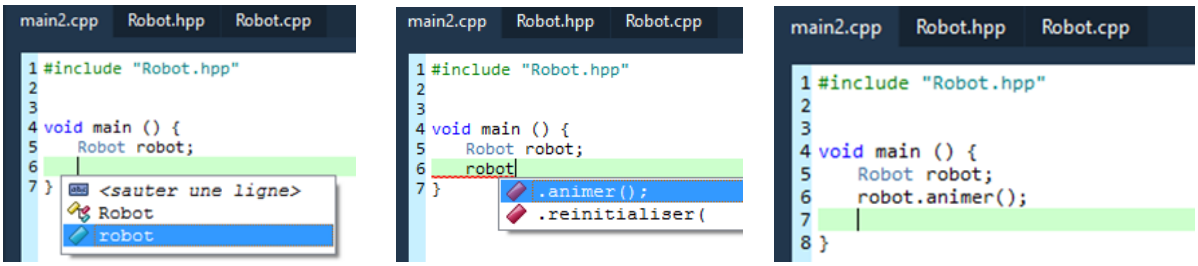


FIGURE 40 – Utilisation d'une nouvelle classe dans PrOgO.

ACCÈS AUX DONNÉES ET FONCTIONS MEMBRES D'UNE CLASSE. Le fichier de déclaration de la nouvelle classe (`*.hpp`) contient l'ensemble de ses données et fonctions membres (attributs et méthodes). Son entête contient les instructions d'inclusion des classes associées aux objets utilisés lors de la constitution de la structure 3D. À titre d'exemple, si la classe nouvellement créée est appelée *Robot* le code de déclaration de l'entête de la classe est comme suit :

```
class Robot {
<...>
};
```

Le corps de la classe *Robot* liste l'ensemble de ses constituants. Ses attributs membres sont les objets utilisés lors de sa construction (Figure 39a). Cette nouvelle classes contient quatre nouvelles fonctions membres, à savoir `initialiserAngles()`, `initialiserCouleurs()`, `animer()` et `reinitialiser()`.

ENCAPSULATION Certaines données et fonctions membres de la classe nouvellement créée sont privées (précédées par le mot clé `private`) et d'autres sont publiques (précédées par le mot clé `public`). Les données membres privées regroupent tous les constituants structurels de la classe. Les fonctions membres comprennent les méthodes `initialiserAngles()` et `initialiserCouleurs()` (Figure 39a). Les fonctions membres publiques consistent en les méthodes `animer()` et `reinitialiser()`. Seules ces deux méthodes sont accessibles depuis la fonction `main()` (Figure 40). Les données et fonctions membres privées sont accessibles uniquement par les autres membres de la même classe (Figure 39b). Ainsi, les fonctions `initialiserAngles()`, et `initialiserCouleurs()` peuvent accéder à tous les attributs membres privés de la classe, mais elles ne peuvent pas être appelées depuis la fonction `main()`. Elles sont en revanche appelées par la fonction `reinitialiser()`. L'utilisateur peut observer ces règles d'encapsulation dans le fichier de définition de sa classe `Robot.cpp` comme dans l'exemple ci-dessous :

```
void Robot::initialiserAngles(){
    engrenage_0.angleDeRotation = 45;
    engrenage_1.angleDeRotation = 0;
```

```
    <...>
}
void Robot::initialiserCouleurs(){
    base.couleur = "#ffffff";
    bifurcationy_0.couleur = "#00ffff";
    bifurcationy_1.couleur = "#ffff11";
    <...>
}
<...>
void Robot::reinitialiser(bool b_angle, bool b_color){
    if (b_angle)
      initialiserAngles();
    if (b_color)
      initialiserCouleurs();
}
```

La méthode `animer()` contient toutes les actions d'animation réalisées préalablement lors de la construction de la nouvelle classe. Le comportement de l'animation peut être modifié dans le corps de la fonction `animer()`. L'utilisateur peut observer le contenu de la méthode `animer()` comme dans l'exemple suivant :

```
void Robot::animer(){
    bifurcationy_0.colorierPendant("#55aa00",10);
    engrenage_1.tournerPendant(90,4);
    <...>
}
```

CONSTRUCTEUR DE CLASSE La classe nouvellement créée possède également un constructeur qui permet d'initialiser un objet lors de sa création (par l'instanciation de cette classe). L'utilisateur peut observer que le constructeur est public, qu'il porte le même nom que sa classe, et qu'il n'a pas de type de retour (Figure 39a). Le constructeur de la classe `Robot` est déclaré comme suit :

```
class Robot {
 <...>
 public:
   Robot();
};
```

Le contenu du constructeur de la nouvelle classe comprend les instructions de connexion des objets qui ont servi à sa construction, ainsi que les méthodes `initialiserAngles() et initialiserCouleurs()` qui initialisent les instances de cette classe à l'état de cette dernière sauvegardé lors de sa création dans le premier mode (Figure 39b). L'utilisateur peut observer cela comme dans l'exemple suivant :

```
Robot::Robot(){
  engrenage_0.connecter(0,base,13);
  engrenage_1.connecter(0,base,9);
  roue_0.connecter(0,engrenage_0,1);
  <...>
```

```
  initialiserAngles();
  initialiserCouleurs();
}
```

5.2.4 *Scénarisation d'activités et génération de traces d'interaction*

PrOgO est conçu de manière à permettre de scénariser des activités d'apprentissage et de fournir un moyen pour la collecte de données afin de préparer des études expérimentales. La scénarisation d'activités consiste essentiellement à donner la possibilité à l'enseignant de décider à quel moment ses étudiants ou élèves peuvent commencer à travailler avec l'interface de PrOgO, et de contrôler leur passage du premier mode (*utilisation d'objets*) au second mode (*création de classes*).

La collecte de données peut être activée ou désactivée selon les besoins de recherches expérimentales que l'enseignant/chercheur souhaite mener. Ces données consistent en les traces d'interaction des étudiants/élèves avec l'interface de PrOgO [4].

Pour ce faire, PrOgO peut être configuré manuellement en passant par la base de registre du système d'exploitation, ou par un utilitaire de configuration indépendant conçu afin de faciliter son paramétrage par un administrateur, un pédagogue ou un enseignant/chercheur (Figure 41). La configuration de PrOgO permet d'activer ou de désactiver les boîtes de dialogue informatives apparaissant lors de l'interaction avec son interface (*cadran 1*), d'activer ou de désactiver la fonctionnalité de collecte de traces d'interaction et de spécifier le chemin de sauvegarde des fichiers de traces dans le cas où cette fonctionnalité est activée (*cadran 2*), et d'activer ou de désactiver la communication réseau et l'administration d'interfaces distantes (*cadran 3*).

Dans le cas où la communication réseau est activée, l'interface de PrOgO sur un poste étudiant/élève devient passive et se retrouve bloquée et mise en attente jusqu'à ce qu'elle soit activée par l'enseignant ou l'administrateur. Au niveau des postes étudiants/élèves, une fenêtre modale apparaît à l'écran au lancement de PrOgO bloquant toute interaction avec ce dernier. L'interaction avec PrOgO devient possible que si l'enseignant ou l'administrateur l'autorise depuis un poste distant. Pour ce faire, un utilitaire indépendant de PrOgO est prévu, offrant une interface de communication avec des dispositifs distants (PC, tablettes, ... etc) connectés au même réseau local. Cette interface permet l'interaction avec les interfaces des étudiants/élèves en classe lors de l'utilisation de PrOgO. Elle offre la possibilité à l'enseignant d'activer ou de désactiver l'interface de PrOgO sur les postes distants des étudiants, d'autoriser le passage du premier mode (*utilisation d'objets*) au second mode (*création de classes*), et de récupérer les travaux des étudiants en les sauvegardant dans un emplacement donné sur le réseau (Figure 42).

4. À noter que PrOgO n'offre aucun moyen de traiter ou d'analyser les traces d'interaction, il se contente uniquement de générer des traces d'interaction brutes.

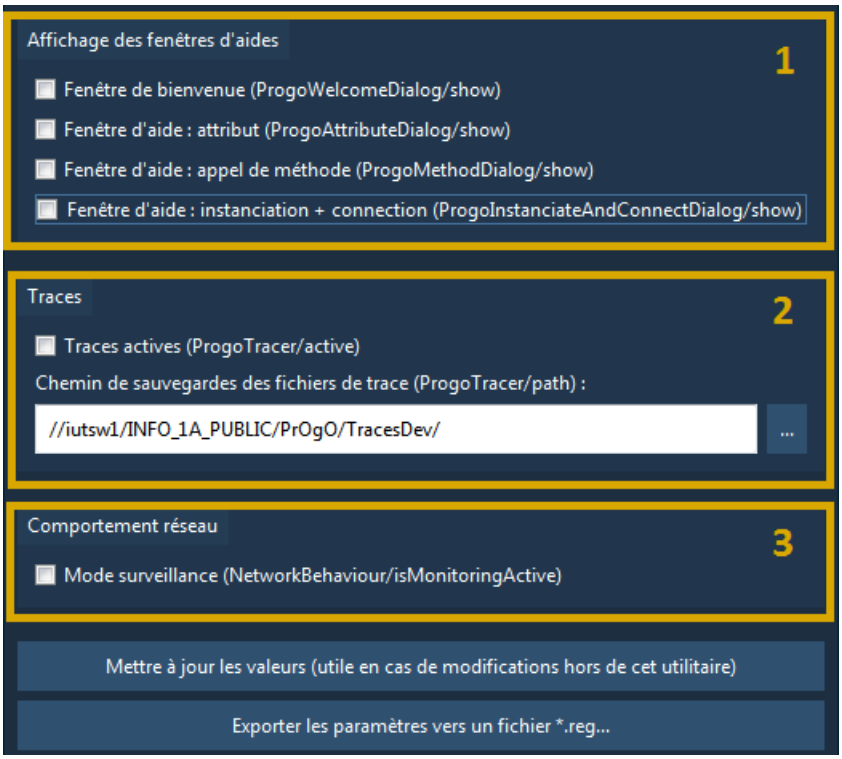


FIGURE 41 – Utilitaire de configuration de PrOgO.

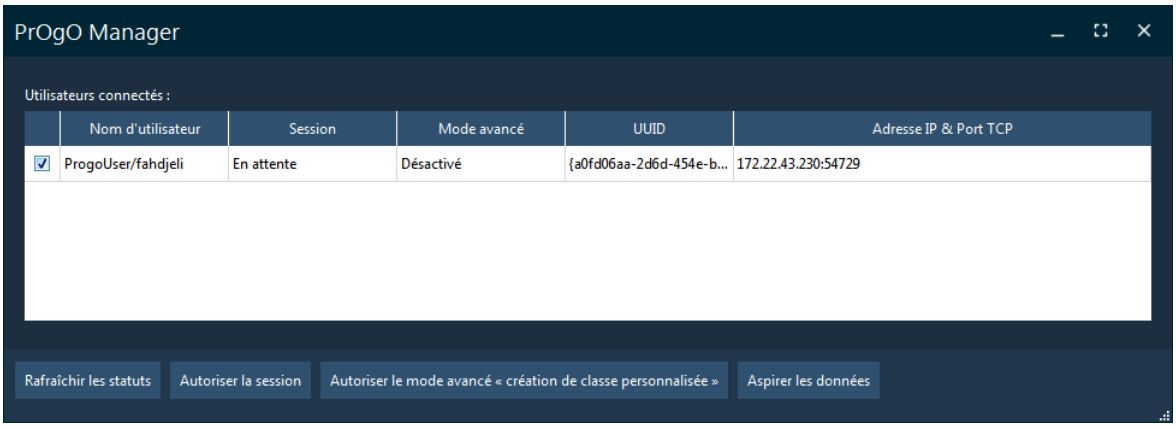


FIGURE 42 – Interface utilisateur de l'utilitaire de gestion des sessions étudiantes sur interfaces distantes.

5.3 HISTORIQUE DE DÉVELOPPEMENT ET ARCHITECTURE GÉNÉRALE

5.3.1 *Développement d'un prototype expérimental et d'un prototype évolutif*

Le développement de PrOgO a débuté en Avril 2014, lors d'un stage de troisième année *Licence Image et Son*, en parcours *3D Temps Réel et Réalité Virtuelle* à l'IUT du Puy en Velay de l'Université d'Auvergne, avec l'implémentation d'un premier prototype expérimental reposant sur Qt comme technologie de développement et Qt 3D comme moteur 3D[5]. Il s'agissait pour l'heure d'une simple interface qui permettait d'utiliser des objets et construire des structures 3D en générant du code C++ qui n'était pas éditable (cf.s. C.1). Le nom "*PrOgO*" a été proposé pour la première fois et a été retenu dans la suite du projet. En Juillet 2014, quand ce prototype a fini d'être implémenté, Qt 3D était peu évolué. Très vite l'idée de l'utiliser comme moteur 3D au sein du projet a été abandonnée.

Dès Février 2015, un prototype évolutif de PrOgO a commencé à être implémenté. Il repose sur Qt comme technologie de développement et Ogre comme moteur de rendu 3D. Ce prototype a évolué principalement de la version V.0.6 (dès Octobre 2015) qui permettait l'utilisation d'objets et l'édition de code C++, vers la version V.0.7 (dès Décembre 2015), qui étendait PrOgO vers le second mode, la création de classes. L'implémentation de PrOgO, dans le cadre du projet *Tactileo*, a pris fin en Avril 2016 avec l'ajout d'utilitaires de configuration de PrOgO et la mise en place de son interfaçage avec le framework Tactileo, dans la prise en compte d'interfaces utilisateurs tactiles et la communication réseau (cf. Chapitre 4).

5.3.2 *Architecture générale*

L'architecture générale de la dernière version stable de PrOgO est donnée sur la Figure 43. PrOgO repose sur les technologie Qt (cf. s. 4.3.1.1) et OGRE[6]. OGRE est un moteur 3D libre et multiplateforme qui s'intègre avec Qt. Il offre une API reposant sur le langage C++, permettant la visualisation et la gestion d'environnements 3D.

L'environnent PrOgO offre une interface graphique qui possède deux composants essentiels en constante interaction, à savoir une scène 3D dont le rendu est géré grâce au moteur 3D OGRE, et un éditeur de code à autocomplétion mis en place grâce aux fonctionnalités offertes par Qt. PrOgO permet la gestion d'entrées utilisateurs tactiles et la découverte automatique du réseau par souscription aux plugins de propagation d'évènements multi-touches et de découverte réseau du framework Tactileo (cf. Chapitre 4). PrOgO permet également de simuler l'exécution pas à pas de code C++ et l'animation de constructions 3D dans la scène, et offre la possibilité de se connecter à *Tactileo Cloud*[7] afin d'effectuer la sauvegarde et le chargement de projets sur le réseau. Enfin, PrOgO

5. http ://doc.qt.io/qt-5/qt3d-index.html
6. http ://www.ogre3d.org/
7. Tactileo Cloud est une plateforme de création et de partage de contenus multi-média interactifs /https://www.tactileo.org/logon. Cette fonctionnalité a été rajoutée sous la demande de la société Maskott.

trace les actions de l'utilisateur qu'il sauvegarde dans un fichier `*.csv` généré pour chaque utilisateur à chaque nouvelle session.

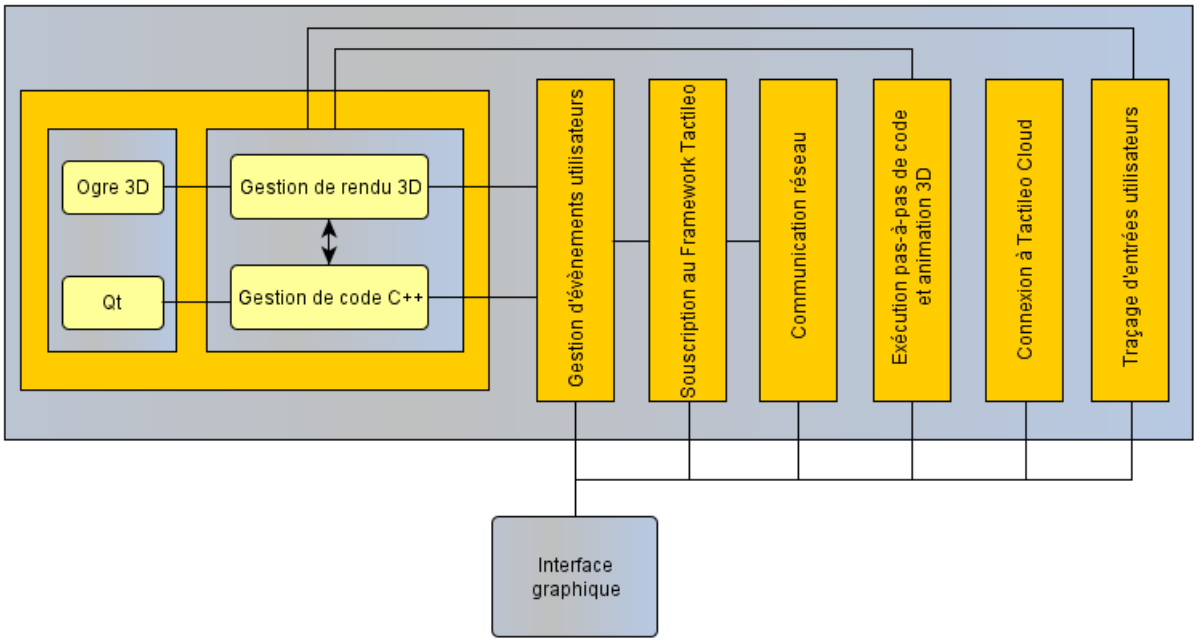


FIGURE 43 – Architecture générale de l'environnement PrOgO.

5.4 DÉTAILS DE CONCEPTION ET D'IMPLÉMENTATION

5.4.1 *Scène 3D interactive*

Afin de visualiser un environnement 3D à l'écran, une scène 3D doit contenir une caméra virtuelle qui doit être positionnée et paramétrée de sorte à ce que son point de vue capte l'environnement que l'on veut visualiser. Pour cela, le point de vue de la caméra (son centre optique ou de projection) est souvent celui de l'utilisateur (l'œil de l'observateur) (Figure 44). Le principe de visualisation comprend essentiellement un *repère du monde*, une *fenêtre du monde*, un *viewport*, un *plan de coupe proche* et un *plan de coupe éloigné*.

Le *repère du monde* est le repère 3D global de la scène. Le terme *monde* est utilisé, car le programme de l'application représente un monde qui se crée de façon interactive ou s'affiche à l'écran de l'utilisateur (D. FOLEY et al., 1995). Pour que les primitive de sortie (les modèles graphiques 3D) soient visualisées, une correspondance doit être établie entre le système de coordonnées du monde et le système de cordonnées de l'écran. Cette correspondance est réalisée en spécifiant une région rectangulaire dans le système de coordonnées du monde appelée la *fenêtre du monde (ou de rendu)*, et une région rectangulaire dans le système de coordonnées de l'écran appelée le *viewport* dans laquelle la fenêtre du rendu est mappée. La *fenêtre du monde (ou de rendu)* délimite la partie du monde que l'on souhaite visualiser grâce à un *plan de coupe proche* (*near clipping plan*) et un *plan de coupe éloigné* (*far clipping plan*). Les objets du monde ne sont

plus rendus à partir du *plan de coupe éloigné*. Afin d'améliorer la visibilité, un point de lumière est ajouté à la scène 3D.

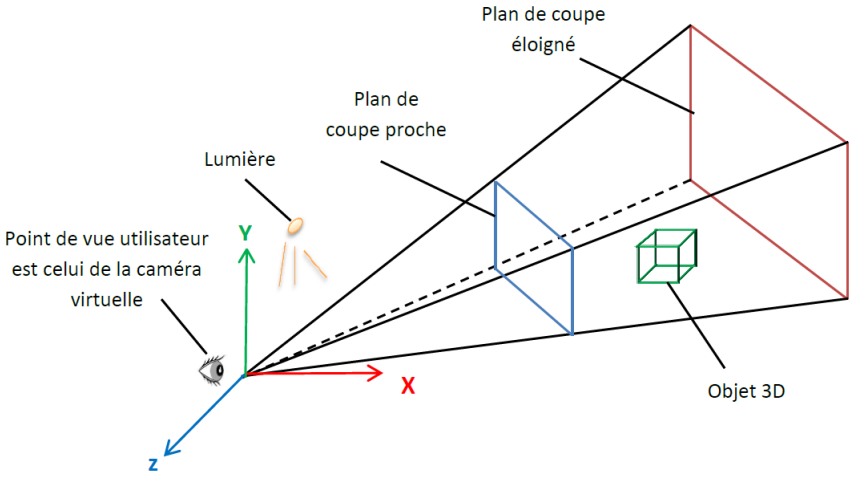


FIGURE 44 – Principe de visualisation 3D.

La scène 3D de l'environnement PrOgO utilise le système de coordonnées propre à Ogre 3D. Ogre utilise le plan (X-Z) comme *sol* de scène. L'axe X commence par des valeurs négatives à gauche qui augmentent vers la droite en passant par l'origine (zéro). Les déplacements le long de l'axe Y se font du bas vers le haut. Les déplacements le long de l'axe Z se font en avant et en arrière, l'axe Z positif pointe hors de l'écran. Ce qui signifie que si un objet 3D se rapproche de l'écran, alors sa coordonnée sur Z augmente.

PrOgO repose sur Ogre dans la gestion du rendu 3D. Ogre gère tout ce qui apparait à l'écran grâce essentiellement à trois classes modélisant trois concepts, à savoir *un gestionnaire de scène, un nœud de scène et une entité* :

— la classe `SceneManager` (gestionnaire de scène) : garde trace des positions et autres attributs des objets présents dans la scène 3D. Elle gère également l'ensemble des caméras ajoutées à la scène.

— la classe `SceneNode` (nœud de scène) : représente un nœud de scène qui n'est pas visible dans la scène. Elle contient les informations relatives à tous les objets attachés à ce nœud de scène. Ces informations sont utilisées uniquement quand un objet, tel qu'une entité, est attaché à ce nœud pour être rendu.

— la classe `Entity` (entité) : est tout objet pouvant être rendu dans la scène 3D représenté par un *mesh 3D* [8]. Les lumières et les caméras sont des exemples d'éléments de scène qui ne *sont pas des entités*. Une entité n'est rendue que si elle est attachée à un nœud de scène.

Tout comme toute application utilisant Ogre, avant d'accéder aux fonctionnalités de ce dernier, PrOgO crée une instance de la classe `Root` de Ogre (*racine*), afin d'initialiser le moteur 3D de Ogre et de spécifier le type de système de rendu à utiliser, ainsi que les chemins d'accès pour le chargement des modèles 3D et de leurs matériaux. Grâce à cet objet racine et aux classes `RenderWindow et SceneManager`, PrOgO crée une fenêtre de

8. Un mesh 3D est un ensemble de polygones élémentaires qui constituent un modèle 3D créé avec un logiciel de modélisation.

rendu et un gestionnaire de scène (Figure 45) en faisant appel aux méthodes respectives `createRenderWindow ()` et `createSceneManager()` de la classe `Root`. Le gestionnaire de scène crée ensuite deux caméras en spécifiant leurs plans de coupe proches et éloignés, et la fenêtre de rendu ajoute deux viewports rattachés à chacune des deux caméras. La première caméra dirige son point de vue vers l'origine du repère 3D de la scène (localisation de la base de construction), servant à visualiser la scène interactive et créative de PrOgO. La seconde caméra fixe son point de vue sur le modèle 3D représentant le repère du monde s'affichant en permanence en bas à gauche de la scène 3D créative. Deux lumières sont également ajoutées à la scène au moyen du gestionnaire de scène afin d'éclairer les objets affichés à l'écran. Enfin grâce à la classe `FrameListener`, l'objet racine enregistre la scène 3D comme étant à l'écoute de notifications d'évènements afin de la rendre interactive.

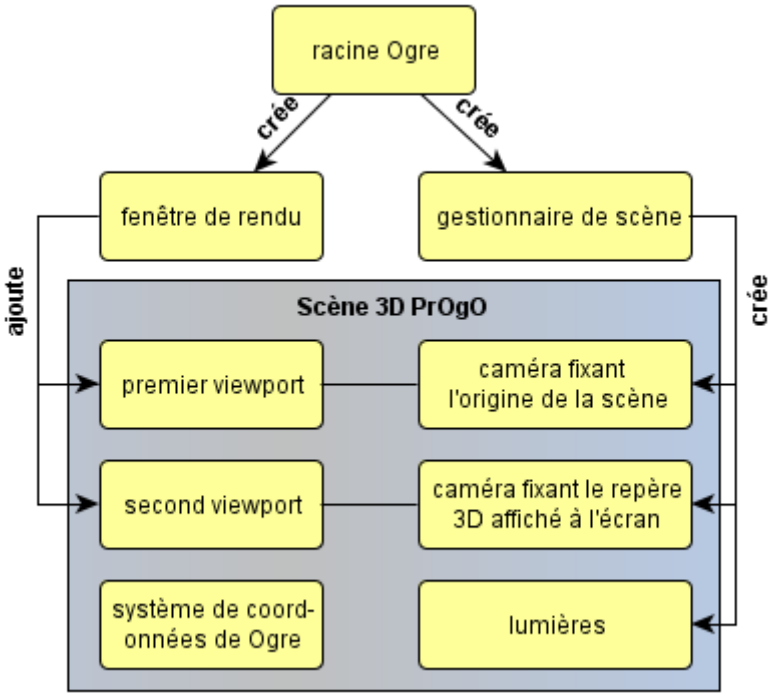


FIGURE 45 – Mise en palace d'un système de rendu dans PrOgO à l'aide de Ogre.

5.4.2 *Objets 3D et connecteurs d'objets*

Au lancement de PrOgO, ou lors de la création ou l'ouverture d'un nouveau projet, l'objet 3D *base* représentatif de la base de construction est visualisé dans la scène 3D dès que le système de rendu 3D est mis en place (création d'une fenêtre de rendu, placement et paramétrage des caméras, des viewports et des lumières, ...etc (cf. s. 5.4.1)).

Tout objet 3D visualisé dans la scène de PrOgO est un nœud de Ogre (instance de la classe `SceneNode` de Ogre). Un objet 3D est visualisé suite à la création puis l'ajout à la scène d'un nœud de scène Ogre auquel est attachée son entité par le biais du gestionnaire de scène. Avant la création d'un nœud de scène, PrOgO récupère son entité ainsi que d'autres informations relatives à son rendu dans un fichier eXtensible

Markup Language (XML) associé à l'objet 3D correspondant. Ce fichier XML est basé sur le fichier d'exportation généré par le logiciel de modélisation *3DsMax*. Les objets 3D de PrOgO sont modélisés avec *3DsMax* puis exportés dans un format compatible avec Ogre à l'aide du module d'exportation *OgreMax*. Dans *3DsMax* chaque objet solide modélisé pour être rendu dans une scène est appelé *géométrie (geometry)*. L'objet modélisé possède une apparence déterminée par son maillage (*mesh*), sa couleur ou son matériau, une position, des valeurs de mise à l'échelle et une rotation dans la scène de modélisation *3DsMax*. Un objets 3D de PrOgO est défini par sa géométrie, position (x,y,z), taille et orientation, et n'a pas de matériau. De plus, chaque objet possède un certain nombre de connecteurs ayant une forme cylindrique possédant le même matériau. Chaque connecteur est placé sur un objet de sorte à ce que son axe Z local soit dirigé hors de l'objet. Lors de l'exportation, un fichier XML est généré par objet 3D modélisé décrivant ses propriétés, incluant sa géométrie et l'emplacement de ses différents connecteurs (Annexe B).

5.4.3 *Techniques d'interaction 3D*

5.4.3.1 *Sélection d'objets dans la scène 3D*

Dans la sélection d'objets 3D, PrOgO implémente la technique du *lancer de rayon au niveau polygonal* (*raycasting to the polygone level*)[9]. Cette technique consiste dans un premier lieu à générer un rayon virtuel depuis le centre de projection de la caméra dirigeant son point de vue vers la scène 3D. Ce rayon traverse le viewport en passant par la position (x,y) du clic souris (ou un contact tactile) normalisée avec les dimensions de l'écran. Ce rayon est retourné grâce à la fonction `getCameraToViewportRay()` de la classe `Camera` de Ogre. Dans un second temps, grâce à la classe `RaySceneQuery` de Ogre, le gestionnaire de scène crée une requête afin de retrouver les points d'intersection du rayon virtuel avec les polygones constituants le mesh de chaque nœud de scène dont la boîte englobante a été intersectée (Figure 46). L'objectif étant de déterminer avec précision quel objet est sélectionné parmi d'autres objets de la scène 3D. Grâce à la classe `RaySceneQueryResultEntry` qui retourne les résultats d'intersection, les distances entre l'origine de projection du rayon et les points d'intersection trouvés sont comparées. Il s'agit alors de déterminer à quel nœud de scène appartient le polygone sur lequel le point d'intersection est le plus proche de l'origine de la caméra.

L'action de sélection d'un objet implique instantanément la surbrillance de son nom dans la hiérarchie d'objets (l'onglet haut gauche de l'interface) et l'apparition de sa boîte englobante. Cette dernière est calculée grâce à la classe `WireBoundingBox` de Ogre et s'affiche à l'écran une fois attachée au nœud de scène correspondant à l'objet sélectionné.

En fonction du type du clic souris (droit ou gauche), du type du contact tactile (pression maintenue ou pas) et du type du nœud sélectionné (l'objet 3D ou l'un de ses connecteurs), PrOgO s'apprête soit à créer un nouvel objet ou à afficher un menu déroulant correspondant au type de l'objet sélectionné (objet structurel ou objet actif).

9. La technique du *raycasting to the polygone level* est documentée sur : `http://www.ogre3d.org/tikiwiki/tiki-index.php?page=Raycasting+to+the+polygon+level`

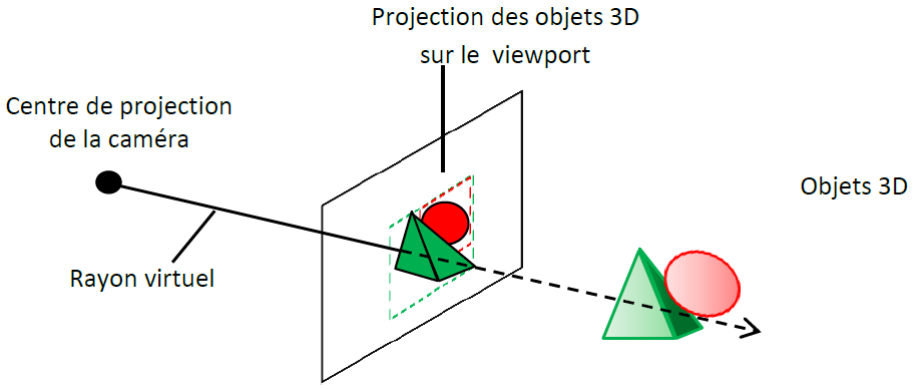


FIGURE 46 – Principe de la technique du lancer de rayon.

5.4.3.2 *Création et assemblage d'objets dans la scène 3D*

Dans PrOgO, chaque action de l'utilisateur peut être annulée ou rétablie. Pour cela, PrOgO se base sur le framework d'annulation de Qt qu'il implémente grâce aux classes `QUndoStack` et `QUndoCommand` fournies dans l'API Qt. Toute nouvelle action de l'utilisateur entraîne la création d'une commande qui est empilée pour être éventuellement annulée ou rétablie.

Au moment où l'utilisateur choisit la classe à instancier dans l'onglet (*classes*), PrOgO sauvegarde le nom de cette dernière. Lorsqu'un nœud de scène est sélectionné par un clic souris gauche (ou un contact tactile) et a été identifié comme un connecteur, PrOgO s'apprête à créer un nouvel objet à partir de ces deux données (le connecteur sélectionné et le nom de la classe choisie). Cela entraîne l'empilement d'une nouvelle commande de création et de connexion d'objets. Le nom de la classe à instancier permet de déterminer le fichier de spécification XML du modèle 3D associé, qui réunit les informations utiles à son rendu (cf. s. 5.4.2).

Avant d'être rendu, l'objet 3D venant d'être créé est assemblé au nœud de scène dont le connecteur vient être sélectionné, en plaçant son premier connecteur ayant pour identifiant `0` à la position du connecteur sélectionné en direction opposée de l'axe Z local de ce dernier (le connecteur étant modélisé en dirigeant son axe Z local hors de l'objet correspondant) (Figure 47). L'objet créé est ensuite ajouté à la scène par le biais du gestionnaire de scène.

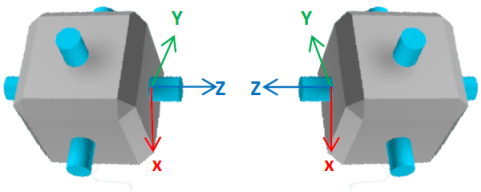


FIGURE 47 – Orientation des connecteurs lors de l'assemblage d'objets.

5.4.3.3 *Manipulation d'objets dans la scène 3D*

Lorsqu'un nœud de scène est sélectionné par un clic souris droit (ou un contact tactile) et a été identifié comme n'étant pas un connecteur, PrOgO affiche un menu

déroulant avec des commandes personnalisées. Les commandes de rotation d'objets ne doivent pas être affichées sur les objets n'ayant pas la capacité de tourner dans la scène. Pour ce faire des informations liées aux propriétés des classes représentées sont manuellement ajoutées aux fichiers de spécification XML associés aux différents modèles 3D (cf. s. 5.4.1) :

```
<classProperties canRotate="bool" canChangeColor="bool"/>
```

Cette ligne XML permet de distinguer les deux catégories de classes (*composants structurels et composant actifs*), et donc d'afficher de manière appropriée les commandes d'un menu déroulant associé au type de l'objet sélectionné. En associant la valeur "`true`" au champ `canRotate`, les commandes de modification de l'attribut `angle` et d'exécution de la méthode `tournerPendant()` sont ajoutées au menu déroulant. De la même manière, si le champ `canChangeColor` possède la valeur "`true`", les commandes de modification de l'attribut `couleur` et d'exécution de la méthode `colorierPendant()` sont affichées.

Chaque action enregistrée entraîne la création et l'empilement d'une nouvelle commande à l'instance de `QUndoStack`, pour être éventuellement annulée ou rétablie. Les commandes de modification de couleurs entraînent la création d'un nouveau matériau qui sera associé à l'entité du nœud de scène correspondant à l'objet sélectionné.

Dans la rotation d'objets, PrOgO utilise la fonction `rotate()` de la classe `Node` de Ogre. L'utilisateur spécifie uniquement la valeur de l'angle de rotation qu'il souhaite. La rotation d'un objet se fait toujours autour de l'axe Z local de son connecteur `0` (axe de connexion) par rapport au repère du monde [10] (Figure 48). La rotation d'un objet entraîne successivement celle de l'objet directement connecté à lui, du fait même de leur connexion.

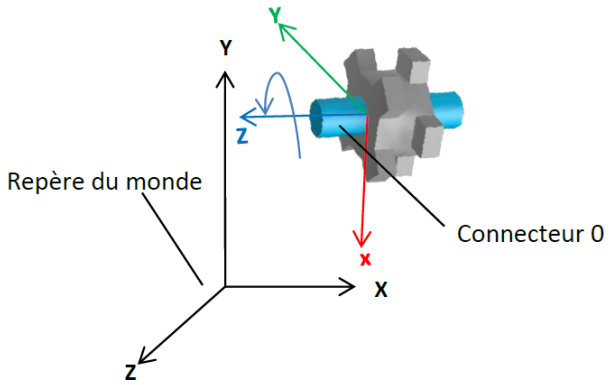


FIGURE 48 – Rotation d'objets dans la scène 3D.

5.4.3.4 *Navigation autour d'une construction 3D*

La navigation dans la scène 3D s'effectue autour de son origine, et se traduit par des opérations de rotation et de replacement de la caméra virtuelle fixant son point de vue sur l'origine de la scène. Deux rotations sont appliquées, la première est réalisée sur l'axe Y, et la seconde sur son axe dérivé droit (Figure 49). La différence entre la position

10. Étant donné que l'objectif final est d'animer la structure 3D, les objets actifs doivent tourner autour de leur axe de connexion. C'est ce qui constitue les différentes articulations d'un robot, par exemple.

(x,y) courante de la mollette de la souris (ou du contact tactile) et sa position précédente enregistrée sur l'écran donne un nouveau point (x,y) dont la coordonnée en X est utilisée pour définir l'angle de rotation de la caméra sur l'axe Y, et la coordonnée en Y définit l'angle de rotation sur l'axe dérivé droit. Ces deux rotations sont suivies du calcul d'une nouvelle position de la caméra en utilisant son vecteur de direction dont la norme est égale à la distance de la caméra de l'origine de la scène. Ces mêmes opérations sont appliquées sur la deuxième caméra présente dans la scène, fixant sont point de vue sur le modèle 3D représentant le repère du monde s'affichant en bas à gauche de la scène 3D [11]. Les opérations de rotation et de positionnement sont respectivement réalisées grâce à la fonction `rotate() et setPosition()` de la classe `Camera` do Ogre.

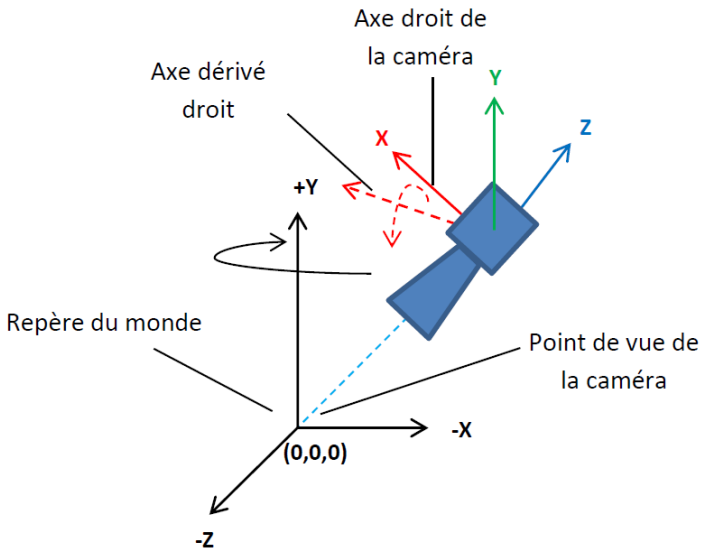


FIGURE 49 – Rotation de la caméra virtuelle dans la scène 3D.

5.4.3.5 *Le zoom dans la scène 3D*

L'action du Zoom permet de se rapprocher ou de s'éloigner de la scène grâce à la molette de la souris (ou un pincement tactile). Cela se traduit par un repositionnement de la caméra virtuelle fixant son point de vue sur l'origine de la scène. En tournant la molette de la souris, la propriété `delta` de l'évènement `wheelEvent` de Qt donne une valeur qui indique de combien la molette souris a été tournée. Cette valeur est négative si la mollette est tournée vers l'utilisateur, et positive si au contraire la molette est tournée vers l'avant. De la même manière, avec un pincement des deux doigts sur l'écran, les positions courantes et précédentes des deux doigts sont utilisées pour calculer un paramètre équivalent au delta de la molette souris. Il s'agit alors d'additionner ou de soustraire au vecteur de position de la caméra, une valeur de zoom par défaut.

11. La rotation du modèle 3D représentant le repère du monde dans la scène de PrOgO, permet à l'utilisateur de s'orienter dans la scène lors de la navigation autour de la construction 3D.

5.4.4 *Autocomplétion de code C++*

L'éditeur de code C++ de PrOgO repose sur la classe `QPlainTextEdit` de Qt, qui fournit un widget pour la visualisation et l'édition de texte claire, et utilise la classe `QCompleter` qui fournit les fonctions d'autocomplétion.

L'éditeur de code met en place un assistant de contenu personnalisé sur les lignes éditables (lignes en surbrillance verte). Les items constituant l'assistant de contenu sont des items standards de QT (de type `QStandardItem` s'utilisant avec `QStandardItemModel`).

L'assistant de contenu est constitué en fonction des zones de code éditable (zone d'entête réservée à l'inclusion de fichiers `*.hpp` et zone de code source). À chaque fois qu'une chaîne de caractères est entrée, elle est analysée afin de mettre à jour l'assistant de contenu et d'afficher à l'utilisateur les options appropriées. À titre d'exemple, quand une chaîne de caractères a été identifiée comme nom d'objet, l'assistant de contenu propose les items de modification d'attributs et d'appels de méthodes en fonction de son type (composant structurel ou composant structurel actif). Cela se fait en vérifiant les valeurs des champs `canRotate` et `canChangeColor` dans le fichier de spécification XML du modèle 3D associé à la classe de cet objet (cf .s. 5.4.3.3).

Dans l'analyse de code (*parsing* de code), PrOgO implémente la technique de *filtrage de motif* (*pattern matching*) en utilisant les expressions régulières (ou rationnelles) grâce aux fonctionnalités de la classe `QRegExp` de Qt (e.g. `exactMatch(), cap(), ...etc.`). Le filtrage de motif consiste à rechercher dans une chaîne de caractères la présence d'un élément qui correspond à un motif spécifique. Un motif est un modèle décrit par une expression régulière, qui est une chaîne de caractères spécifiant le type et l'ordre des caractères constituant le motif (Julien et Grandval, 2015 ; Mettier, 2014).

PrOgO définit un motif pour chaque instruction C++ utilisée en respectant le langage des expressions régulières de Qt comme suit :

— instructions d'inclusion de fichiers de déclaration de classes :

```
#include\\s\"(\\S+).hpp\"\\n
```

— instructions d'instanciation de classes :

```
(\\S+)\\s(\\S+);
```

— instructions de modification d'attribut :

```
(\\S+)\\.(\\w+)\\s?=\\s?(\\S+)\\s?;
```

— instructions d'appel de méthodes :

```
(\\S+)\\.(\\w+)\\((.*)\\);
```

À noter que la barre oblique inversée précède tout caractère recherché, les deux parenthèses délimite un sous-motif, `\s` correspond au caractère espace, `\S` correspond à tout caractère différent du caractère espace,`\w` correspond à tout caractère alphanumérique (comprend aussi le caractère '_', ou de catégorie *Mark_ de Qt*), `\n` indique le passage à une nouvelle ligne, `+,* et ?` indiquent respectivement une ou plusieurs occurrences, aucune ou plusieurs occurrences et une occurrence ou plus du caractère précédant.

Les commandes de saut de ligne et d'invitation de saisie de paramètres à l'aide de boîtes de dialogue sont identifiées grâce ou motif suivant : `<(.*)>`

À chaque fois qu'une instruction C++ est entrée par autocomplétion, l'analyseur de code lance un filtrage de motif afin de créer de manière appropriée les commandes de création et d'assemblage d'objets 3D ou de manipulation d'objets 3D. Une instruction d'instanciation d'une classe entraîne le chargement du fichier XML du modèle 3D associé (cf. s. 5.4.2), et implique la création et l'ajout d'un nœud de scène Ogre par le biais du gestionnaire de scène. Une instruction de connexion spécifiant les objets à connecter et leurs connecteurs associés entraîne la création et l'empilement d'une commande de création et de connexion d'objets, de la même manière qu'un assemblage d'objets dans la scène 3D (cf. s. 5.4.3.2). Les instructions de manipulation d'objets (modification de valeur d'attribut ou appel de méthode) entraînent la création et l'empilement des commandes de manipulation de la même manière que dans la scène 3D (le résultat d'exécution d'une instruction est visible dans la scène 3D) (cf. s. 5.4.3.3).

5.4.5 *Exécution pas à pas de code et animation 3D*

PrOgO implémente un simulateur d'exécution pas à pas de code et d'animation 3D. Pour ce faire, il parcourt le programme principal dans l'éditeur de code ligne par ligne afin de constituer une file d'exécution d'instructions. Cette file contient l'ensemble des instructions préalablement générées ou entrée par autocomplétion, incluant les différentes variables définies, les fonctions appelées et les paramètres des fonctions. Chaque pas d'exécution entraîne un changement dans la vue 3D avec une valeur de 100 `FPS` (images par seconde). L'algorithme du simulateur interroge la file d'instructions toutes les 10 millisecondes afin de les exécuter dans l'ordre. Dès qu'une instruction est exécutée, elle est supprimée de la file.

L'animation est engendrée par l'exécution des instructions `colorierPendant()` et `tournerPendant()`. En effet, ces instructions s'exécutent pendant une durée de temps spécifiée par l'utilisateur. Il s'agit pour PrOgO de changer la couleur du matériau du nœud en question ou de changer son angle de rotation dans la scène 3D (cf. s. 5.4.3.3). Avec une valeur `FPS` égale à 100, le simulateur génère une image toutes les 10 millisecondes, en calculant un nombre de fois, soit une nouvelle couleur ou une nouvelle rotation assignée à l'objet correspondant jusqu'à ce que le temps spécifié par l'utilisateur soit écoulé. Pour chacun des attributs couleur et angle de rotation, la nouvelle valeur est calculée en interpolant la valeur courante de l'attribut et la valeur finale spécifiée par l'utilisateur. Le pas d'interpolation est donné par le rapport entre le temps écoulé depuis le début de l'exécution de l'instruction et le temps total d'exécution de l'instruction.

5.4.6 *Gestion des entrées tactiles*

Comme pour les évènements souris, Qt capte automatiquement un point de toucher dès la première pression à l'intérieur d'un composant graphique, et le composant graphique recevra toutes les mises à jour du point de toucher jusqu'à ce que ce dernier soit relâché. Qt offre aussi un mécanisme qui permet à un composant graphique de recevoir à la fois des évènements souris (`QMouseEvent`) et des évènements de touchers (`QTouchEvent`) pour un même point d'interaction avec l'utilisateur. De ce fait, maintenir

un doigt appuyé sur un écran tactile effectue la même action qu'un clic souris droit, et une pression du doigt non maintenue est équivalente à un clic souris gauche.

Pour recevoir des évènements tactiles, l'interface graphique de PrOgO fixe l'attribut `Qt::WA_AcceptTouchEvents` comme requis par Qt. Afin de gérer les gestes de déplacement et de pincement des doigts, PrOgO utilise le plugin de propagation d'évènements multi-touches et tangibles du framework Tactileo (cf. s. 4.3.4.1). Dans un premier temps, PrOgO inclue les interfaces `inputspligininterface, inputevent, et cursorinterface` qui elle-même inclue `contactinterface` du gestionnaire d'évènements du framework (cf. s. A.1). Une instance du plugin est alors créée après que ce dernier ait été localisé et chargé grâce à `QPluginLoader` de Qt (cf. s. 4.3.4). PrOgO souscrit ensuite aux évènements tactiles en appelant la fonction `Subscribe()` du plugin, ayant pour premier paramètre le composant de son interface devant répondre aux évènements tactiles, et comme second paramètre le flag `ListenNativeCursorEvents`.

Une fois la souscription effectuée, PrOgO récupère facilement la liste des contacts tactiles (objets `Cursor` de *TUIO*) grâce à la méthode `getAllAliveCursors()`. Un contact tactile est traité grâce aux fonctionnalités de l'interface `ContactInterface` (e.g. getPositions(), getState()), et les informations relatives à son état sont récupérées grâce à l'interface `InputEvent` (`CURSOR_ADDED, CURSOR_UPDATED, CURSOR_REMOVED`) (cf. s. A.1).

Tout changement intervenu sur un contact tactile est détecté grâce à l'évènement `CURSOR_UPDATED`. Cela concerne :

— le mouvement de déplacement d'un doigt unique sur l'écran dans la scène 3D, qui entraîne la navigation autour de l'origine de la scène. La différence entre la position courante du contact du doigt et sa position précédente sur l'écran est utilisée pour tourner les caméras (cf. s. 5.4.3.4).

— le mouvement de pincement des deux doigts sur l'écran dans la scène 3D, qui entraîne le rapprochement ou l'éloignement de la scène (*zoom*). La différence entre le vecteur des positions courantes des deux doigts et le vecteur de leurs positions précédentes sur l'écran est utilisée pour repositionner la caméra fixant son point de vue sur l'origine de la scène (cf. s. 5.4.3.5).

5.4.7 *Communication réseau et paramétrage d'interfaces utilisateurs distantes*

La fonctionnalité de communication réseau et de paramétrage d'interfaces utilisateurs distantes est mise en place à la fois dans l'utilitaire de gestion des communications réseaux et dans PrOgO. Ils utilisent tous les deux le plugin de découverte automatique du réseau du framework Tactileo (cf. s. 4.3.4.2) : dans un premier temps, les interfaces `networkpluginginterface, networkdeviceevent et networkdeviceinterface` sont inclues (cf. s. A.2). Une instance du plugin est alors créée après que ce dernier ait été localisé et chargé grâce à `QPluginLoader` de Qt. PrOgO souscrit ensuite au réseau en tant que *dispositif racine* grâce à la méthode `SubscribeAsDevice()`, tandis que l'utilitaire souscrit en tant que *dispositif racine* et *point de contrôle* de façon simultanée via la méthode `SubscribeAsControlPointAndDevice()` (cf. s. 4.3.4.2).

Au lancement, l'utilitaire de gestion des communications réseau se met à l'écoute d'évènements de présence (`DEVICE_ADDED et DEVICE_REMOVED`) de dispositifs distants (cf. s. 4.3.3.2). Ces évènements sont émis de manière transparente par toutes les instances

de l'application PrOgO s'exécutant sur des terminaux connectés au même réseau local (PrOgO souscrit au réseau en tant que *dispositif racine*). À chaque fois qu'un événement `DEVICE_ADDED` est émis, l'utilitaire détermine le dispositif émetteur de l'évènement (en faisant appel à la méthode `getDevice()` de l'interface `NetworkDeviceEvent`) et l'identifie grâce aux fonctionnalités de l'interface `NetworkDeviceInterface` (eg. `getUUID()`). Cette interface permet également de récupérer des informations de connectivité du dispositif connecté, telles que son adresse Internet Protocol (IP) V4, son serveur Transmission Control Protocol (TCP) et le numéro de port de ce dernier (en faisant appel aux méthodes `getIPV4Address () getTcpServerInstance() et getTcpServerPort()`). Les informations relatives aux dispositifs détectés et identifiés sont alors affichées sur l'interface graphique de l'utilitaire afin que l'utilisateur (l'enseignant) choisisse les interfaces de PrOgO à paramétrer qui sont en cours d'exécution sur les terminaux identifiés.

La communication entre l'utilitaire de gestion réseau et l'interface de PrOgO connectée au même réseau local se fait par envoi de messages TCP. À chaque fois qu'un nouveau paramétrage est réalisé sur l'interface graphique de l'utilitaire, un socket TCP est envoyé au terminal réseau sur lequel PrOgO est en cours d'exécution. Ce socket est intercepté et traité par PrOgO afin d'exécuter la commande lancée depuis le terminal de l'utilitaire. PrOgO envoie à son tour des réponses TCP à l'utilitaire afin que celui-ci mette à jour les informations mises à disposition de l'utilisateur sur son interface graphique, quant à l'état de l'interface de PrOgO sur le terminal distant.

5.5 FORCES ET LIMITES DE PROGO

PrOgO est un micromonde de POO qui offre à l'apprenant un espace de créativité et d'imagination dans une situation de jeu-play. Les forces de PrOgO résident dans ses fondements conceptuels, c'est-à-dire :

— une métaphore de construction et d'animation *fidèle et puissante* dans la description des notions fondamentales de la POO. Elle est aussi *consistante* car elle permet la description de systèmes OO complexes, comprenant les relations d'interaction entre classes.

— des *objets transitionnels* illustrant la métaphore de construction et d'animation, au travers de graphiques 3D interactifs et visibles à l'interface. Leur comportement est semblable à celui de concepts fondamentaux de la POO. L'objectif étant d'aider l'apprenant à produire du sens sur les concepts formels de la POO, en manipulant de façon intuitive des objets concrets à l'interface.

— un *jeu-play* qui peut se mettre en place quand l'apprenant accepte d'interagir avec l'interface. L'objectif étant de jouer à un jeu de construction et d'animation 3D, en vue de mettre en place l'apprentissage des concepts fondamentaux de la POO.

— une *approche didactique par emboîtement hiérarchique* des concepts fondamentaux de la POO, ayant pour objectif un apprentissage progressif de notions très fortement interliées. L'emboîtement des concepts fondamentaux de la POO au travers des modes *utilisation d'objets* et *création de classes*, a pour objectif de permettre à l'apprenant de se concentrer sur les notions liées à l'*objet*, tout en occultant les détails liés à la notion de *classe*. Ces dernières sont volontairement occultées dans le premier mode, pour être introduites en détail dans le second mode.

Les limites de PrOgO concernent d'abord des fonctionnalités qui n'ont pas été mises en place dans la limite du temps alloué à son développement au cours du projet Tactileo. Ces limites sont essentiellement les suivantes :

— PrOgO n'implémente pas le *mode concepts* de l'approche didactique par emboîtement hiérarchique des fondamentaux de la POO. La version 0.7 se limite aux deux premiers modes de l'approche, *utilisation d'objets* et *création de classes*. L'interface de PrOgO occulte dans le second mode, les notions liées aux aspects conceptuels de l'OO tels que les relations hiérarchiques entre classes. Le troisième mode est donc manquant.

— Le modèle OO décrivant les constructions robotiques 3D est unique pour toutes les réalisations des apprenants. Or, la description d'exemples différents de systèmes OO, est en mesure d'enrichir le modèle de connaissances de PrOgO. Cela peut aider davantage l'apprenant dans la production de sens sur le paradigme OO, en manipulant des exemples multiples de systèmes OO différemment modélisés.

Certaines limites concernent les fonctionnalités d'interaction utilisateur, telles que :

— Lors de la construction 3D, l'interface de PrOgO n'offre pas la fonctionnalité de translation dans la scène 3D. Les objets 3D sont figés autour de l'origine de la scène, à la position de la base de construction.

— les calculs de détection des collisions ne sont pas traités, ce qui produit quelquefois un effet visuel qui ne reproduit pas fidèlement le comportement réel d'objets solides.

Enfin, il est important de souligner certaines limites concernant l'éditeur à auto-complétion :

— Le code généré suite aux interactions directes des objets 3D dans la scène virtuelle, devient très vite répétitif. Ce qui conduit à la surcharge de l'éditeur avec du code qui perd son utilité et qui peut ennuyer l'apprenant.

— PrOgO ne possède pas une API propre, et le code possible à entrer en autocomplétion se restreint à la syntaxe liée à l'objet et à la classe.

— l'éditeur à auto-complétion se limite à analyser une syntaxe restreinte et n'est pas fondé sur un vrai compilateur.

5.6 CONCLUSIONS ET RÉSUMÉ DES CONTRIBUTIONS

PrOgO est un micromonde de POO fondé sur un jeu de construction et d'animation 3D. Il a été conçu sur la base d'autres micromondes existants et se fonde sur des concepts que nous avons pu ressortir de notre recherche bibliographique portée par trois grandes questions :

— Qu'est ce que le jeu et quelle est sa relation à l'apprentissage ? (Chapitre 1)

— Quelle approche pour l'enseignement introductif de la POO ? (Chapitre 2)

— Quels artefacts informatiques pour l'introduction de la POO par le jeu ? (Chapitre 3)

A ces questions de recherche, s'ajoute une question de contexte, qui est la classe *Tactileo* :

— Comment prendre en compte les exigences de la classe *Tactileo*, pour que PrOgO puisse s'intégrer dans une classe multi-dispositifs tactile ? (Chapitre 4)

Ces questions ont conduit aux fondements conceptuels de PrOgO, à savoir :

— un *jeu-play* qui peut se mettre en place quand l'apprenant accepte d'interagir avec PrOgO. Le jeu-play consiste à construire et à animer des structures 3D.
— une *approche didactique par emboîtement hiérarchique* des concepts fondamentaux de la POO. Cette approche vise un apprentissage progressif de notions hiérarchiques très fortement liées les unes aux autres.
— une *métaphore de construction et d'animation fidèle, puissante et consistante* dans la description des concepts fondamentaux de la POO.
— un *système de représentation transitionnel*, dans lequel la métaphore de construction et d'animation illustre, au travers de graphiques 3D interactifs, les concepts de POO. Les graphiques 3D sont des objets transitionnels, car leur comportement est semblable à celui des concepts fondamentaux de la POO. L'interaction avec ces objets transitionnels est en mesure d'aider l'apprenant à produire du sens sur les concepts formels de la POO.
— gestion d'*entrées tactiles* et *communication réseau* avec d'autres applications connectées au même réseau dans la classe *Tactileo*.
— génération de *traces numériques d'interaction* à des fins expérimentales.

De part ses fondements conceptuels issus de nos recherches, PrOgO contribue à l'état de l'art des micromondes existants. Il implémente des principes conceptuels propres aux micromonde de programmation, et permet l'étude expérimentale de ses usages par la génération de traces d'interaction numériques.

Par ailleurs, PrOgO présente plusieurs limites, notamment ses fonctionnalités d'édition de code. L'éditeur de code se limite à des fonctionnalités d'autocomplétion et repose sur l'analyse d'une syntaxe C++ restreinte, qui se limite aux notions d'objet et de classe. PrOgO présente néanmoins, une base de travail très riche pouvant nourrir la recherche sur les micromondes de programmation existants.

Dans le chapitre suivant, nous présentons nos études expérimentales menées avec PrOgO visant à explorer et à démontrer son potentiel sur l'apprentissage des fondamentaux de l'OO.

6

ÉVALUATION DES USAGES DE PROGO ET DIAGNOSTIC DES CONNAISSANCES D'APPRENANTS

6.1 INTRODUCTION

L'objectif pédagogique de PrOgO est d'aider les étudiants débutants à comprendre les concepts fondamentaux de la POO de façon rapide et intuitive. Le développement de PrOgO est passée par trois étapes essentielles donnant lieu à un premier prototype expérimental, puis à un second prototype qui a évolué de la version 0.6 à la version 0.7. Une étude expérimentale a été menée sur chacun des trois prototypes, afin d'étudier le potentiel de PrOgO dans l'introduction des fondamentaux de la POO aux débutants.

Ce chapitre a pour objectif de décrire les différentes expérimentations réalisées sur les différents prototypes de PrOgO. Il présente chaque étude expérimentale en décrivant ses objectifs, la méthodologie employée dans la collecte et l'analyse des données dans les différents contextes d'expérimentation, et les résultats d'analyse ainsi que et les conclusions tirées de leurs discussions.

La première expérimentation a été consacré à l'évaluation des ressentis des apprenants et de leurs sentiments d'avoir appris avec le prototype expérimental de PrOgO, à l'aide d'un questionnaire (Section 6.2). La seconde expérimentation a principalement pour objectif d'étudier les comportements d'apprenants sur le prototype de PrOgO v.0.6, par l'analyse de traces numériques d'interaction (Section 6.3). La troisième expérimentation a pour objectif de vérifier l'acquisition des connaissances chez les apprenants débutants (Section 6.4) en utilisant PrOgO v. 0.7. Cette étude se base sur une évaluation pré-test/post-test des connaissances et sur l'analyse de traces numériques d'interaction, visant à expliquer les facteurs déterminant la progression dans l'apprentissage. Enfin, ce chapitre conclut sur nos principales contributions (Section 6.5).

6.2 ÉVALUATION DES RESSENTIS ET DES PERCEPTIONS SUBJECTIVES AVEC LE PROTOTYPE EXPÉRIMENTAL DE PROGO

6.2.1 *Objectif de l'expérimentation*

L'objectif de l'expérimentation menée avec le prototype expérimental de PrOgO (cf.s. C.1) consiste à valider nos choix de conception liés aux aspects ergonomiques, ludiques et pédagogiques de PrOgO, en s'appuyant sur les jugements subjectifs et opinions d'étudiants débutants. Ceci a pour but d'établir une base de conception d'un prototype *évolutif* de PrOgO (Djelil, B. Albouy-Kissi et al., 2015).

6.2.2 *Méthodologie*

6.2.2.1 *Contexte de l'expérimentation*

Tous les ans, au sein de l'IUT du Puy en Velay (Université d'Auvergne), les cours introductifs de la programmation se font en première année *DUT Imagerie Numérique* lors du premier semestre, dans le cadre de l'unité d'enseignement *Bases de l'informatique*, qui comprend les modules "*Introduction à l'algorithmique et à la programmation*" et "*Structures de données et algorithmes fondamentaux*". Le premier module couvre les bases de la programmation impérative en langage C++ : *types de données, variables, opérateurs, structures de contrôle conditionnelles et itératives, tableaux, pointeurs, procédures et fonctions*. Le second module est une introduction aux bases de la POO en langage C++ : *classes, objets, encapsulation, polymorphisme, surcharge d'opérateurs et relations entres classes comprenant l'agrégation, la composition et l'héritage*.

L'expérimentation a été réalisée en Octobre 2014 auprès d'étudiants de première année universitaire en *DUT Imagerie Numérique à l'IUT du Puy en Velay*. Les étudiants ont achevé le premier module (*Introduction à l'algorithmique et à la programmation*) et ont tout juste entamé le second (*Structures de données et algorithmes fondamentaux*). Ils ont donc été préalablement introduits pour la première fois à quelques concepts élémentaires de la POO, tels que *la classe, l'objet, l'encapsulation et l'invocation de méthodes*. Les participants étaient au total 49 et leur âge moyen était de 18.7 ans. Ils ont tous affirmé, qu'ils jouent aux jeux vidéo très fréquemment (31 participants jouent tous les jours et 18 d'entre eux jouent plusieurs fois par semaine). Ils ont également majoritairement rapporté ne pas jouer aux jeux sérieux (37 participants ne jouent jamais contre 12 participants qui jouent régulièrement).

Pendant 1h20 les participants étaient amenés à donner libre court à leur imagination pour construire des structures 3D avec le prototype expérimental de PrOgO. Ensuite, ils ont répondu à un questionnaire sur leurs points de vue pendant 20mn (cf. s. F.4) .

6.2.2.2 *Collecte de données*

Nous avons recueilli les appréciations subjectives des participants au moyen d'un questionnaire basé sur leur expérience d'interaction avec ce premier prototype de PrOgO. Les participants étaient amenés à exprimer leurs sentiments d'avoir appris et leurs appréciations des aspects ludiques du prototype. Ils ont également exprimé leurs opinions quant à l'utilisation future de PrOgO, en vue d'apprendre d'autres concepts complexes fondés sur les concepts considérés. Enfin, les participants ont donné leur avis quant aux aspects conceptuels qu'ils souhaitaient voir s'améliorer.

Nous avons collecté les traces d'interaction des étudiants, afin d'examiner leurs actions et de vérifier si ces derniers faisaient l'effort de comprendre le lien entre les graphiques 3D et le code généré suite à leurs actions dans la scène 3D, et ne pas se focaliser uniquement sur cette dernière. Les traces d'interaction enregistrées comportent des actions incluant la création/destruction d'objets et l'invocation de méthodes dans la scène 3D, ainsi que la sélection d'instructions C++ dans les onglets de génération de code. Nous avons également réuni les réalisations graphiques 3D des participants afin de vérifier si les étudiants étaient impliqués dans le *jeu-play*.

6.2.3 *Résultats et discussion*

Les opinions des participants recueillies grâce au questionnaire, sont exprimées sur une échelle à quatre niveaux (*pas du tout*, *peu*, *plutôt*, *beaucoup*). Nous avons mis en relation le sentiment d'amusement des participants à la fréquence à laquelle ils jouent au jeux vidéos. De la même manière, leur sentiment d'avoir amélioré leur compréhension des concepts abordés est mis en relation avec la fréquence à laquelle ils jouent aux jeux sérieux. Les résultats obtenus sont ensuite comparés (Tableau 7).

Item de comparaison	participants	Nombre (%)	*pas du tout* (%)	*peu* (%)	*plutôt* (%)	*beaucoup* (%)
PrOgO est-il amusant ?	étudiants jouant aux jeux vidéos tous les jours	63.27	10	51	39	0
	étudiants jouant aux jeux vidéos plusieurs fois par semaine	36.73	17	39	44	0
	tous	100	12	47	41	0
PrOgO est-il pédagogique ?	étudiants jouant aux jeux sérieux	24.50	0	34	58	8
	étudiants ne jouant pas aux jeux sérieux	75.50	3	22	67	8
	tous	100	2	25	65	8
combien estimez-vous avoir amélioré votre compréhension du concept de classe ?	étudiants pensant avoir préalablement compris en classe	36.73	16	44	36	4
	étudiants pensant ne pas (ou peu) avoir préalablement compris en classe	63.27	8	59	33	0
	tous	100	12	51	35	2
combien estimez-vous avoir amélioré votre compréhension du concept d'objet ?	étudiants pensant avoir préalablement compris en classe	38.78	26	37	26	11
	étudiants pensant ne pas (ou peu) avoir préalablement compris en classe	61.22	3	54	43	0
	tous	100	12	47	37	4
combien estimez-vous avoir amélioré votre compréhension de la syntaxe liée aux concepts de classe et d'objet ?	étudiants pensant avoir préalablement compris en classe	35.42	39	33	28	0
	étudiants pensant ne pas (ou peu) avoir préalablement compris en classe	64.58	3	62	32	3
	tous	100	16	51	31	2

Tableau 7 – Ressentis et points de vue d'étudiants.

Nous avons déduit que les étudiants ont majoritairement trouvé PrOgO *peu* ou *plutôt* amusant et *plutôt* pédagogique, indépendamment du fait qu'ils jouent aux jeux vidéos *tous les jours* ou *plusieurs fois par semaine*. Ce résultat est également indépendant du fait *qu'ils jouent ou pas* aux jeux sérieux (Tableau 7). En effet, les résultats obtenus ne sont pas très différents pour ces deux catégories de participants : 51% des étudiants jouant au jeux vidéos *tous les jours* et 39% de ceux jouant *plusieurs fois par semaine* ont rapporté qu'ils ont trouvé PrOgO *peu amusant*. 39% de ceux jouant *tous les jours* et 44% de ceux jouant *plusieurs fois par semaine* l'ont trouvé *plutôt amusant*. Ils sont au total 47% à le trouver *peu amusant* et 41% à le trouver *plutôt amusant*.

Aucuns des étudiants jouant aux jeux sérieux ne l'a trouvé *pas du tout pédagogique*, ils sont en revanche 58% à penser qu'il est *plutôt pédagogique* et 34% à penser qu'il est *peu pédagogique*. Ceux ne jouant pas aux jeux sérieux l'ont majoritairement trouvé *plutôt pédagogique* (67% contre 22% à l'avoir trouvé *peu pédagogique*).

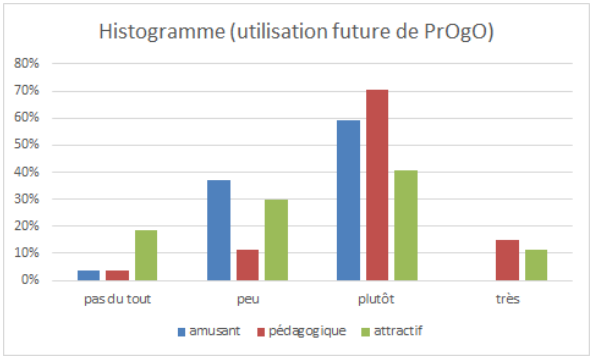
D'autre part, nous avons déduit que les étudiants ont majoritairement eu le sentiment d'avoir *peu* ou *plutôt* amélioré leur compréhension des concepts considérés (la classe, l'objet et la syntaxe C++ liée à ces concepts), indépendamment du fait qu'ils pensent avoir préalablement compris en classe ou pas (Tableau 7). Les étudiants ayant affirmé avoir préalablement compris ces concepts en classe, ont rapporté qu'ils ont quand même amélioré leur compréhension (44% ont eu le sentiment d'avoir *peu amélioré* leur compréhension du concept de *classe*, 37% ont affirmé pareillement pour le concept d'*objet* et 33% pour la syntaxe de ces concepts). Un grand nombre d'entre eux pensent avoir *plutôt* amélioré leur compréhension (36% concernant le concept de classe, 26% concernant le concept d'objet et 28% concernant la syntaxe de classe et d'objet). Les étudiants ayant affirmé ne pas avoir préalablement compris en classe ont également enregistré des pourcentages similaires (59% ont eu le sentiment d'avoir *peu amélioré* leur compréhension du concept de classe. Les pourcentages enregistrés pour le concept d'objet et la syntaxe de classe et d'objet sont respectivement 54% et 62%). Un grand nombre d'entre eux ont rapporté avoir *plutôt amélioré* leur compréhension de ces concepts, leurs pourcentages respectifs sont (33%, 43% et 32%).

On peut observer également que les étudiants ayant affirmé avoir compris en classe ont enregistré des pourcentages significatifs en rapportant ne *pas du tout* amélioré leur apprentissage avec PrOgO (16% pour le concept de classe, 26% pour le concept d'objet et 39% pour la syntaxe de ces concepts). Cela peut indiquer que les étudiants éprouvent plus de difficultés à comprendre le sens des concepts plutôt que leur syntaxe associée. De plus un pourcentage significatif des étudiants (11%) ont dit avoir *beaucoup* amélioré leur compréhension des *objets*, bien qu'ils pensent avoir préalablement compris. Ceci est un résultat qui doit être pris en compte, car le prototype expérimental de PrOgO se concentre sur les objets et les présentent de manière plus concrète que les méthodes magistrales employées en classe.

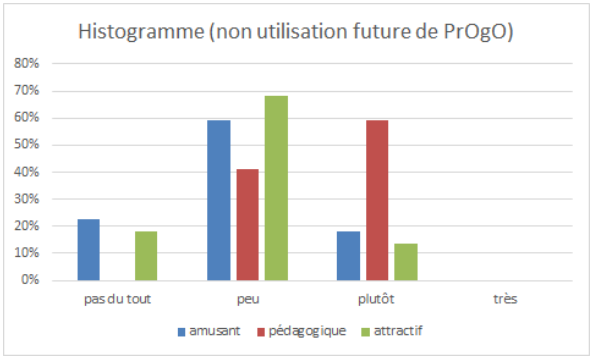
Les opinions des étudiants quant à l'utilisation future de PrOgO sont mises en relation avec leurs jugements subjectifs concernant le fait qu'ils aient trouvé PrOgO *amusant, pédagogique et attractif*. Globalement, les étudiants souhaitant une utilisation future de PrOgO sont ceux qui l'ont trouvé *plutôt* amusant, *plutôt* pédagogique et *plutôt* attractif. Ceux ne souhaitant pas une utilisation future de PrOgO, sont ceux qui l'ont trouvé *peu* amusant, *peu* pédagogique et *peu* attractif (Figure 50).

Les aspects du jeu que les étudiants souhaitaient voir s'améliorer concernent des aspects en lien avec l'apprentissage et l'ergonomie de l'interface. Un grand nombre d'étudiants ont souhaité d'avantage de graphiques 3D et plus d'options comme la possibilité de manipuler simultanément des groupes d'objets. La plupart des étudiants ont exprimé leur intérêt pour modifier le code généré afin de pouvoir interagir plus facilement avec la scène 3D.

Nous avons observé dans les traces d'interaction, que les étudiants étaient majoritairement focalisés sur la manipulation des graphiques dans la scène 3D. Les actions les plus réalisées par les étudiants sont la création d'objets et les appels de méthodes sur les objets dans la scène 3D, à savoir *assembler, déplacer et pivoter* (Tableau 8). Certains étudiants étaient également intéressés par le code généré, étant donné que les traces comportaient les actions "*sélectionner code*" et "*focus onglet*" (focus sur l'onglet de code).



(a) Étudiants souhaitant une utilisation future.



(b) Étudiants ne souhaitant pas une utilisation future.

FIGURE 50 – Utilisation future de PrOgO en lien avec les jugements des étudiants.

action	NB actions [1]
instancier	1868
assembler	4915
déplacer	4051
pivoter	3710
colorier	969
désassembler	118
focus onglet [2]	190
sélectionner code	582

[1] Nombre d'actions pour l'ensemble des participants.

[2] Focus mis sur l'onglet de code.

Tableau 8 – Actions des participants quantifiées à partir de leurs traces d'interaction.

Grâce aux réalisations graphiques des étudiants, on peut observer que certains d'entre eux ont clairement réussi à construire des structures 3D significatives et amusantes (Figure 51). Cela indique que les étudiants étaient impliqués et déterminés dans la réalisation de l'objectif du *jeu-play*.

6.2.4 *Conclusions*

À partir de ces résultats et de leur discussion, nous avons conclu que bien que PrOgO soit au stade de prototype, les étudiants ont majoritairement eu le sentiment d'avoir amélioré leur compréhension des concepts abordés et de s'être amusés. De plus, plus de la moitié d'entre eux ont souhaité une future utilisation de PrOgO afin de découvrir d'autres concepts. Un grand nombre d'entre eux ont souhaité avoir plus d'options ergonomiques de l'interface, ainsi que la possibilité de modifier le code généré, chose que nous avions en perspectives. Les traces d'interaction ont montré que les étudiants ont tendance à se focaliser sur l'interface 3D, mais certaines actions ont été enregistrées dans l'onglet de code qui n'était pas encore éditable.

Bien que ce travail soit une étude préliminaire, ces résultats étaient encourageants. On est arrivé à la conclusion qu'une version stable de PrOgO pouvait être en mesure d'aider les débutants à apprendre les fondamentaux de la POO. Cette étude nous a

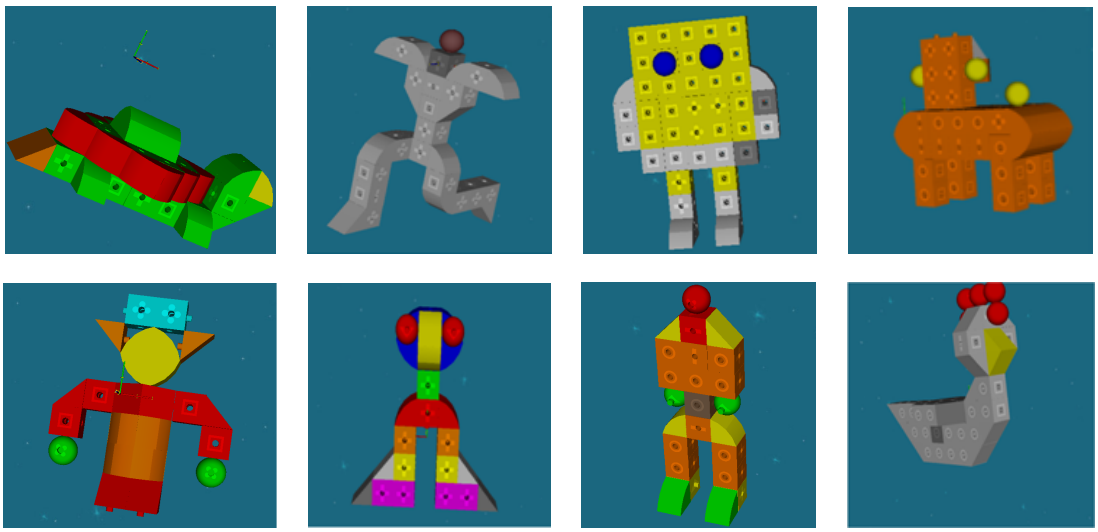


FIGURE 51 – Quelques réalisations 3D d'étudiants.

permis de valider nos choix conceptuels, notamment la *métaphore de la construction* sur laquelle PrOgO est fondé. Un grand nombre d'étudiants ont affirmé avoir amélioré leur compréhension de la notion d'objet, représentée visuellement par des objets graphiques 3D illustrant cette métaphore. Les réactions des étudiants nous ont permis de confirmer les extensions que nous prévoyions pour un nouveau prototype évolutif.

Par ailleurs, à ce stade de prototype, PrOgO n'offrait pas la possibilité d'éditer le code généré suite aux actions de l'utilisateur dans l'interface 3D. Par conséquent, une analyse plus fine des traces d'interaction n'était pas envisagée et nos résultats basés sur l'observation de leur contenu n'étaient pas très significatifs. De même, notre méthodologie d'évaluation qui s'est concentré sur le recueil des appréciations et des opinions des participants a montré ses limites. En effet, les résultats obtenus pour les participants concernant leurs sentiments d'avoir appris et de s'être amusés, étaient indépendants de la fréquence à laquelle ils jouent aux jeux vidéos, leur expérience avec les jeux sérieux et leur sentiment d'avoir (ou pas) préalablement compris en classe. Dans la section suivante nous présentons une étude empirique plus complète basée sur une analyse statistique des traces d'interaction du prototype évolutif de PrOgO, en vue d'analyser les comportements d'étudiants et de déduire des indicateurs d'utilité et d'acceptabilité de ce micromonde. Une analyse statistique de scores obtenus à un test de vérification de connaissances a été également réalisée.

6.3 ANALYSE DES USAGES PAR LES TRACES D'INTERACTION, VÉRIFICATION DE L'ÉTAT DES CONNAISSANCES ET RECUEIL DES APPRÉCIATIONS SUBJECTIVES AVEC PROGO V.0.6

6.3.1 *Objectifs et questions visées*

Notre première motivation pour cette étude est d'évaluer l'usage de PrOgO, afin d'élaborer une typologie des comportements d'étudiants débutants et de comprendre

comment il est adopté par ces derniers. Cela peut conduire à des indicateurs d'utilité, d'utilisabilité et d'acceptabilité en lien avec l'acquisition des connaissances, l'intérêt et la motivation des étudiants. Un deuxième objectif est d'estimer le potentiel de PrOgO dans l'introduction des concepts fondamentaux de la POO aux débutants. Notre intention n'est pas d'élaborer une évaluation complète des implications de PrOgO sur l'acquisition des connaissances, l'objectif est plus modeste : il s'agit de vérifier l'état des connaissances des étudiants, introduits pour la première fois à l'OO à l'aide de PrOgO. Nous considérons trois questions essentielles : *premièrement, quelles sont les attitudes adoptées par les étudiants lors de leur interaction avec PrOgO ? Deuxièmement, quel est l'état des connaissances des étudiants après utilisation de PrOgO ? Enfin, PrOgO, est-t-il attrayant et engageant du point de vue des étudiants ?*

Une première hypothèse en lien avec la première question concerne les attitudes adoptées par les participants. Ces derniers devraient passer du temps sur l'interface de PrOgO et devraient manipuler la plupart des concepts représentés, à la fois dans la scène 3D et dans l'éditeur de code. Cela devrait les aider à comprendre le lien modélisé entre les concepts abstraits et formels de la POO et les représentations graphiques de ces concepts à l'interface. Par conséquent, les participants devraient être capables d'analyser le paradigme OO et maîtriser rapidement les concepts représentés. Une seconde hypothèse est en lien avec les résultats d'analyse des scores des participants obtenus à un test de vérification de connaissances. Ils devraient être capables de maîtriser de manière efficace les concepts d'*objet, classe, attributs, méthodes et état d'objet*. Enfin, ils devraient être déterminés et intéressés dans l'utilisation de PrOgO. Plus ils passent du temps sur l'interface, plus ils sont susceptibles de produire du sens à partir de leur manipulation de l'interface.

6.3.2 *Méthodologie*

Dans la réponse à la première question : quelles sont les attitudes adoptées par les étudiants lors de leur interaction avec PrOgO ? Nous avons conduit une analyse statistique des actions des participants au travers de traces d'interaction avec l'interface de PrOgO. Cette partie de la méthodologie relève du domaine des *Learning Analytics* (cf. s. 1.4.4). Afin d'arriver à élaborer une typologie de comportements d'étudiants et d'identifier l'ensemble des actions determinant leurs similarités et dissimilarités, on s'est appuyé sur les méthodes d'analyse statistique Analyse en Composantes Principales (ACP) (cf. s. G.1) de type *pearson(n)* (Jolliffe, 2002) et Classification Ascendante Hiérarchique (CAH) (cf. s. G.2) (Day et Edelsbrunner, 1984), appliquées à des données constituées à partir des traces d'interaction. Cette analyse a été réalisée sur le logiciel *XLSAT* [1] qui s'utilise avec *Excel*.

Dans la réponse à la seconde question : quel est l'état des connaissances des étudiants après utilisation de PrOgO ? Nous avons élaboré un test de vérification des connaissances en nous basant sur la *Taxonomie de Bloom Révisée* (Anderson et Krathwohl, 2001). Des paramètres statistiques appliqués aux scores des étudiants obtenus à ce test sont calculés, afin de vérifier la qualité du test.

1. https://www.xlstat.com

Dans la réponse à la troisième question : PrOgO, est-t-il attrayant et engageant du point de vue des étudiants ? Nous avons examiné les jugements subjectifs des participants, concernant leurs sentiments de plaisir et de détermination.

6.3.2.1 *Contexte de l'expérimentation*

L'expérimentation a été réalisée en Octobre 2015 auprès d'étudiants débutants issus de deux niveaux d'études distincts. Des étudiants de première année universitaire en *DUT Imagerie Numérique à l'IUT du Puy en Velay* (Université d'Auvergne) et des élèves de *Terminale en filière Science et Technologie au lycée Charles et Adrien Du Puy* (Le Puy en Velay).

Les participants étaient au total 67, dont 55 étudiants universitaires et 12 lycéens. Les étudiants universitaires ont achevé le premier module (*Introduction à l'algorithmique et à la programmation*) et n'ont pas encore entamé le second (*Structures de données et algorithmes fondamentaux*) (cf. s. 6.2.2.1). Au lycée *Charles et Adrien Du Puy*, les élèves ont des connaissances élémentaires en programmation impérative en langage C++ : *types de données, variables, opérateurs, structures de contrôle conditionnelles et itératives*. L'ensemble des participants n'étaient donc pas préalablement initiés à la POO.

L'expérimentation a duré 2h. Pendant 1h30 les participants avaient à disposition un tutoriel d'utilisation décrivant l'interface de PrOgO v.0.6 (cf.s. C.2). Le tutoriel explique également l'introduction des concepts OO à travers l'interface, à la fois dans la scène 3D et dans l'éditeur de code. Les participants étaient amenés à donner libre court à leur imagination pour construire et animer des robots ou des structures mécaniques 3D. Durant cette expérimentation les traces d'interaction des participants avec l'interface de PrOgO ont été sauvegardées dans des fichiers externes (*.csv). La génération de traces d'interaction a été réalisée directement par PrOgO v.0.6. Il s'agit de sauvegarder pour chaque participant l'historique des évènements de son interaction avec l'interface de PrOgO, dans un ordre chronologique de ses actions (Annexe D). Ensuite, les participants ont répondu à un test de vérification de connaissances pendant 30 mn, durant lesquelles ils ont également été amenés à exprimer leurs sentiments de divertissement et de détermination dans la réalisation de l'objectif du *jeu-play*. Le test de vérification de connaissances était accompagné de deux exemples de programmes en langages C++. Un généré par PrOgO et un autre comportant des écritures générales en C++.

6.3.2.2 *Collecte et analyse de données*

COLLECTE ET ANALYSE DE TRACES NUMÉRIQUES D'INTERACTION. Les traces d'interaction consistent en les actions séquentielles des participants avec l'interface de PrOgO, telles que les actions liées à la création d'objets dans la scène 3D, la modification de valeurs d'attributs d'objets et les appels de méthodes sur les objets. Les traces incluent également les actions réalisées dans l'éditeur à autocomplétion, comme la sélection et la complétion de code.

Une fois les traces brutes collectées, elles sont traitées afin de constituer des nouvelles données d'analyse. Les fichiers de traces brutes (*.csv) sont fusionnés en un fichier unique réunissant l'ensemble des actions des participants.

Huit variables d'analyse sont constituées, donnant pour chaque participant, le nombre d'actions réalisées sur l'interface de PrOgO et le temps total passé sur cette interface durant l'expérimentation (Tableau 9). Les actions réalisées dans la scène 3D sont quantifiées et exprimées en *nombre d'actions d'instanciation de classes et de connexion d'objets, nombre d'actions de modification de valeurs d'attributs et nombre d'appels de méthodes*. Les actions réalisées dans l'éditeur de code sont exprimées en *nombre d'actions de sélection de code et en nombre d'actions de complétion de code*.

Variable	Signification
NB instancier-connecter	Nombre d'actions d'instanciation de classes et de connexion d'objets dans la scène 3D
NB setAttributCouleur	Nombre d'action de modification de la valeur d'attribut `couleur` dans la scène 3D
NB setAttributAngleRotation	Nombre d'actions de modification de la valeur d'attribut `angleDeRotation` dans la scène 3D
NB aplFoncColorierPend	Nombre d'appels de la méthode `colorierPendant()` dans la scène 3D
NB aplFoncTournerPend	Nombre d'appels de la méthode `tournerPendant()` dans la scène 3D
NB sélectionCode	Nombre d'actions de sélection de code
NB complétionCode	Nombre d'actions de complétion de code
Temps total	Temps total passé sur l'interface de PrOgO durant l'expérimentation

Tableau 9 – Variables d'analyse de l'ACP.

L'ensemble des participants désignés par leurs identifiants constituent *les observations* de l'ACP. L'objectif étant d'identifier et de visualiser les *corrélations* existantes entre les variables d'analyse et les axes de l'ACP retenus, afin d'arriver à caractériser de manière significative les différents participants.

Afin d'arriver à notre but final qui est d'identifier et de visualiser des groupes *uniformes et distincts* d'étudiants/élèves, nous avons réalisé une CAH (Day et Edelsbrunner, 1984) sur les coordonnées des observations dans le sous-espace vectoriel constitué par les axes de l'ACP retenus. Après cela, nous avons appliqué une seconde fois une ACP sur les mêmes variables d'analyse initiales auxquelles nous avons rajouté, comme *variable qualitative supplémentaire, le groupe auquel appartient chaque observation*, tel que retourné par la CAH. L'objectif étant de caractériser de manière significative les individus formant les différents groupes.

La combinaison de l'ACP et de la CAH nous a permis d'obtenir un graphe bidimensionnel de l'ensemble des observations colorées selon le groupe auquel elles appartiennent. Afin de complémenter les résultats visuels de ce *graphe des observations*, nous avons examiné les coordonnées des *objets centraux* des différents groupes sur les axes de l'ACP retenus. Les objets centraux représentent *les observations les plus atypiques*. Ils ont des caractéristiques très similaires aux autres observations du même groupe, tout en étant très distincts des observations appartenant à d'autres groupes. De ce fait, les objets centraux constituent une aide supplémentaire dans l'identification complète des tendances au sein d'un même groupe.

ÉLABORATION D'UN TEST DE CONNAISSANCES ET ANALYSE DES SCORES. Dans l'élaboration d'un test de vérification de connaissances, nous nous sommes appuyés sur les trois niveaux inférieurs de la taxonomie de Bloom révisée des objectifs d'apprentissage (Annexe E).

Lors de cette expérimentation, les objectifs d'apprentissage visés par PrOgO v 0.6 concernent des concepts de base élémentaires indispensables pour comprendre d'autres concepts en lien avec des systèmes OO plus complexes. Les étudiants sont amenés à *utiliser des objets*, en vue de maîtriser des concepts inhérents au concept d'*objet*, que nous rappelons comme suit :

— *Le lien existant entre l'objet et la classe* : un objet est une instance de classe.
— *Les caractéristiques d'un objet* : un objet se caractérise par des attributs et des méthodes.
— *La modification de l'état d'un objet* : modifier la valeur d'un attribut d'objet ou réaliser un appel de méthode d'objet, provoquera la modification de l'état de cet objet.

PrOgO est conçu de façon à introduire l'ensemble de ces concepts au travers de son interface et d'amener l'étudiant à les implémenter en langage de programmation C++. Par conséquent, le test doit pouvoir estimer en terme de scores, après utilisation de PrOgO, la capacité des étudiants à :

— *identifier et nommer* ces différents concepts,
— *expliquer* certains éléments en lien avec ces concepts,
— *implémenter* ces concepts en langage de programmation C++.

Les objectifs du test sont en concordance avec les trois premiers niveaux hiérarchiques de la taxonomie de Bloom révisée : *Reconnaître, Comprendre et Appliquer*. Cela nous amène à construire notre test de vérification de connaissances sur la base de ces trois premiers niveaux d'objectifs d'apprentissage. Le test est composé de 18 questions de différents types : 5 questions à réponses courtes, 7 questions à réponses longues, 4 questions à choix unique et deux questions à choix multiples (cf. s. F.1). Le Tableau 10 résume les objectifs des différentes questions constituant ce test.

Niveau de la taxonomie	Question
Niveau 1 : *Reconnaître*	- Lister des objets par leur noms dans un programme édité dans PrOgO
	- Identifier pour chaque objet listé sa classe correspondante
	- Faire correspondre une syntaxe donnée à un changement de valeur d'attribut
	- Faire correspondre une syntaxe donnée à un appel de méthode
	- Faire correspondre une syntaxe donnée à un appel de méthode en identifiant les objets appelants
	- Identifier pour chaque objet les instructions qui modifient son état
Niveau 2 : *Comprendre*	- Conclure sur le lien existant entre l'objet et sa class
	- Conclure sur le rôle du nom d'objet
	- Conclure sur les rôles des attributs et méthodes d'objet
	- Conclure sur la modification de l'état d'objet
	- Interpréter à partir d'écritures générales les instructions d'instanciation de classes
	- Interpréter à partir d'écritures générales les instructions de changement de valeurs d'attributs
	- Interpréter à partir d'écritures générales les instructions d'appels de méthodes
Niveau 3 : *Appliquer*	- Implémenter une instanciation de classe
	- Implémenter une modification de valeur d'attribut
	- Implémenter un appel de méthode

Tableau 10 – Description des questions constituant le test de vérification des connaissances.

Afin d'analyser de manière significative les scores obtenus à ce test et d'estimer sa qualité, trois paramètres statistiques sont calculés, à savoir un *indice de difficulté*, un *indice de discrimination* et un *indice de fiabilité* (cf.s. G.3).

6.3.3 *Résultats*

6.3.3.1 *Typologie des comportements des étudiants/élèves*

RÉSULTATS DE L'ACP. L'ACP a retourné huit facteurs (composantes principales ou axes) calculés à partir des variables d'analyse listées précédemment (Tableau 9). Les *valeurs propres* des trois premiers facteurs ont permis de cumuler 61.45% de la *variabilité* initiale des données (Tableau 11). Nous avons retenu les trois premiers facteurs et ignoré les cinq autres. Le facteur F4 possède une valeur propre quasiment égale à celle du facteur F3 et les valeurs propres des autres facteurs sont insignifiantes. En effet, il convient de retenir un nombre restreint de facteurs ayant cumulé un maximum de variabilité (cf. s. G.1.2).

	F1	**F2**	**F3**	F4	F5	F6	F7	F8
Valeur propre	**2.36**	**1.48**	**1.08**	1.04	0.79	0.73	0.30	0.22
Variabilité (%)	**29.45**	**18.54**	**13.45**	13.03	9.89	9.13	3.69	2.80
% cumulé	**29.45**	**47.99**	**61.45**	74.48	84.37	93.50	97.20	100

En gras sont les facteurs retenus avec leurs valeurs propres, variabilités et variabilités cumulées exprimées en (%).

Tableau 11 – Facteurs (composantes principales ou axes) retournés par l'ACP.

Le cercle des corrélations constitué sur les axes (F1,F2) montre que l'axe F1 est lié (fortement corrélé) aux variables `NB instancier-connecter`, `NB SélectionCode et NB ComplétionCode` (Figure 52). En effet, ces variables maximisent leurs coordonnées sur l'axe F1 (en positif ou en négatif) et minimisent leurs coordonnées sur l'axe F2. En revanche, l'axe F2 est fortement corrélé avec la variable `Temps total`. Cette dernière possède une coordonnée s'approchant de +1 sur l'axe F2, tout en minimisant sa deuxième coordonnée sur l'axe F1.

Variables	F1	F2	F3
NB instancier-connecter	**-0.75**	0.18	0.01
NB setAttributCouleur	-0.43	0.46	-0.17
NB setAttributAngleRotation	-0.24	0.37	-0.01
NB aplFoncColorierPend	0.39	0.11	**0.66**
NB aplFoncTournerPend	0.32	0.36	**0.62**
NB sélectionCode	**0.80**	0.31	-0.32
NB complétionCode	**0.79**	0.29	-0.34
Temps total	-0.18	**0.90**	-0.03

Les valeurs en gras correspondent aux corrélations les plus fortes entre les variables et les facteurs.

Tableau 12 – Corrélations des variables d'analyse avec les axes F1, F2 et F3.

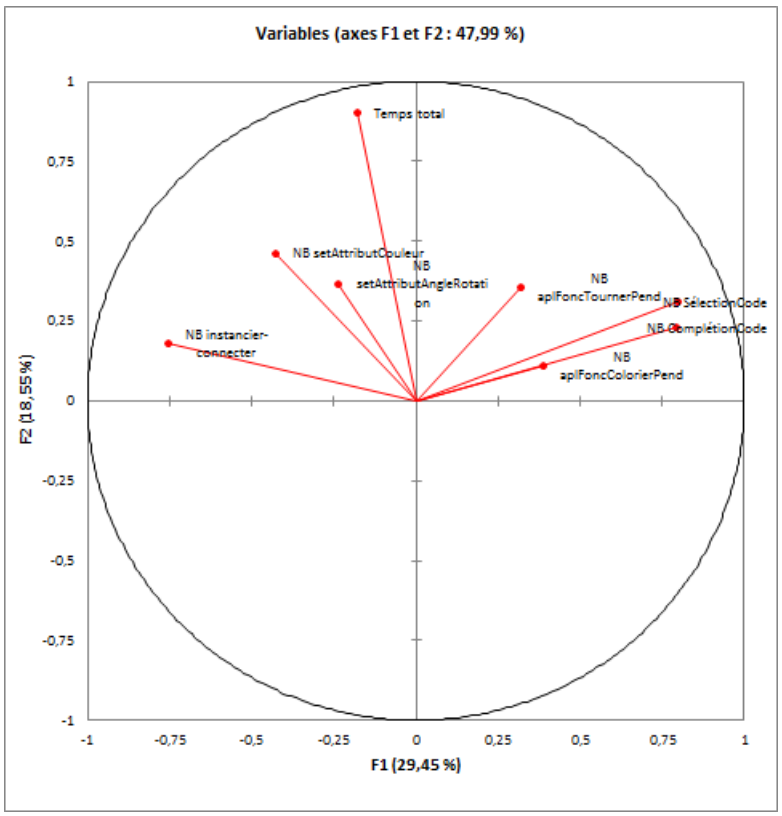


FIGURE 52 – Cercle des corrélations montrant les variables définissant les axes F1 et F2.

Les autres variables se rapprochant du centre du cercle des corrélations, ne servent pas à définir les axes F1 et F2. En effet, en regardant le cercle des corrélations [2] constitué sur les axes (F1,F3), on s'aperçoit que les variables fortement corrélées avec l'axe F3 sont `NB aplFoncColorierPend et NB aplFoncTournerPend` (Figure 53). Ces deux variables maximisent leurs coordonnées sur F3 et minimisent leurs coordonnées sur F1.

Le Tableau 12 peut confirmer les variables définissant les trois premiers axes retenus. En effet, les variables les plus fortement corrélées avec l'axe F1 sont `NB instancier-connecter, NB sélectionCode et NB complétionCode`, ayant pour valeurs de corrélation respectives $-0,75$, $0,80$ et $0,79$ avec l'axe F1. L'axe F2 est positivement corrélé avec la variable `Temps total`. La valeur de corrélation entre F2 et `Temps total` est de $0,90$. Enfin, les variables définissant l'axe F3 sont `NB aplFoncColorierPend et NB aplFoncTournerPend`, leurs valeurs de corrélation étant maximales avec cet axe, respectivement $0,66$ et 0.62.

À partir de ce premier résultat, on peut déduire que les participants se distinguent par ce qui suit :

1. le nombre d'actions réalisées dans l'éditeur de code et le nombre d'actions de création et de connection d'objets dans la scène 3D (variables ayant défini l'axe F1).
2. le temps total passé sur l'interface de PrOgO (variable ayant défini l'axe F2).

2. La corrélation est mesurée par le *coefficient de Pearson* qui prend ses valeurs dans l'intervalle $[-1,+1]$.

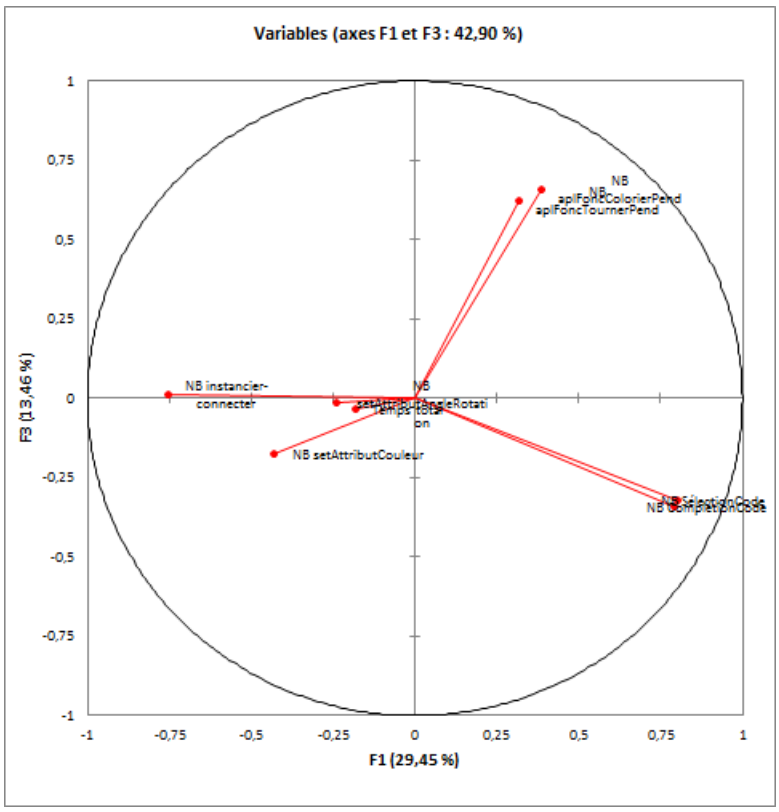


FIGURE 53 – Cercle des corrélations montrant les variables définissant les axes F1 et F3.

3. le nombre d'actions d'animation des réalisations 3D (variables ayant défini l'axe F3).

Cette caractérisation conduit à la déduction des tendances de l'ensemble des participants visualisés sur le plan (F1,F2) ou (F1,F3). Le graphe des observations (F1,F2) (Figure 54) montre que certains individus ont tendance à se concentrer sur les actions de création dans la scène 3D et à négliger l'autocomplétion de code. En effet, ces individus sont projetés sur le semi-plan négatif par rapport à l'axe F1, qui est à la fois corrélé négativement avec la variable `NB instancier-connecter` et positivement avec les variables `NB sélectionCode et NB complétionCode`. En revanche, certains participants sont entièrement concentrés sur l'éditeur à autocomplétion. On voit en effet que ces individus sont projetés sur le semi-plan positif par rapport à l'axe F1. On remarquera particulièrement l'individu E14 qui maximise sa coordonnée sur l'axe F1. Il s'agit de l'étudiant qui a enregistré un nombre maximal d'actions de complétion de code. En effet, les données initiales montrent que ce participant a enregistré 313 actions de complétion de code. Le Tableau 13 confirme qu'il s'agit du maximum d'actions de complétion de code enregistré pour l'ensemble des participants.

On voit également dans le graphe des observations (F1,F2) qu'un grand nombre de participants sont projetés dans le semi-plan positif par rapport à l'axe F2. Cela signifie qu'ils ont passé du temps sur l'interface de PrOgO, la variable de temps étant positivement corrélé avec l'axe F2. On peut particulièrement remarquer le participant E27 qui est projeté sur l'extrémité négative de l'axe F2, maximisant la valeur absolue de sa coordonnée sur cet axe. Il s'agit de l'étudiant ayant passé le moins de temps sur l'interface

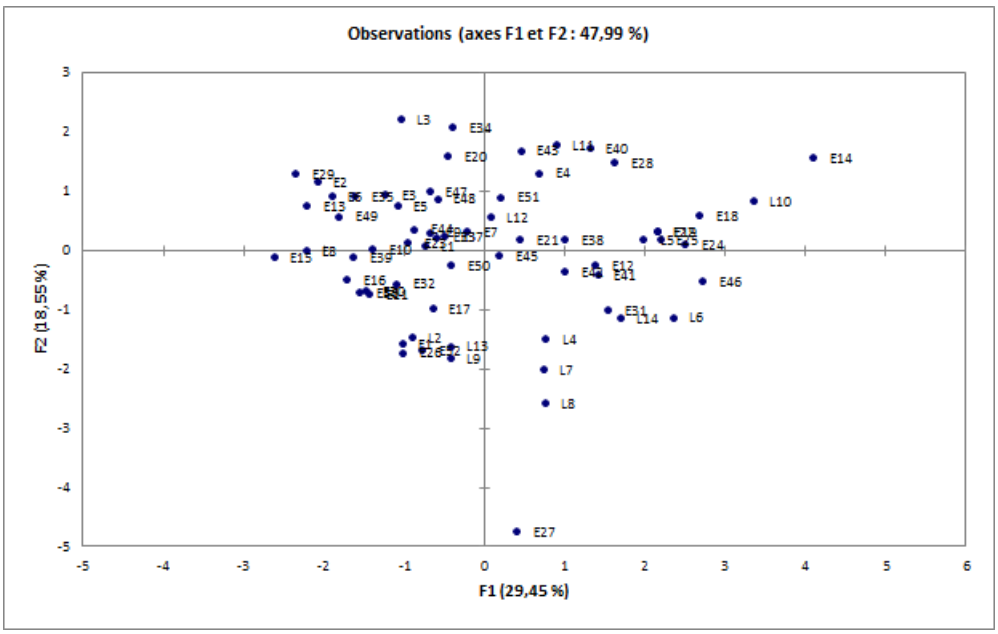


FIGURE 54 – Graphe des observations (F1,F2).

Variables	Minimum	Maximum	Moyenne
NB instancier-connecter	1	646	234.62
NB setAttributCouleur	0	235	56.27
NB setAttributAngleRotation	0	64	7.80
NB aplFoncColorierPend	0	44	2.47
NB aplFoncTournerPend	0	28	6.29
NB sélectionCode	0	31	10.11
NB complétionCode	0	313	43.97
Temps total	0.008 [1]	0,067 [2]	0.051 [3]

[1] 0 :11 :40

[2] 1 :37 :07

[3] 1 :12 :53

Tableau 13 – Statistiques descriptives de l'ACP pour l'ensemble des participants.

de PrOgO. En effet, en regardant les données initiales de l'ACP, on s'aperçoit que le participant E27 a passé 11mn40s. sur le temps total alloué à l'expérimentation qui est de 1h30mn. Le Tableau 13 confirme qu'il s'agit de la valeur minimale de la variable `Temps total` pour l'ensemble des participants.

On remarquera sur le graphe des observations (F1,F3) (Figure 55), que certains individus se caractérisent particulièrement par un nombre important d'action d'animation (variables définissant l'axe F3), tels que E24, L10 et E28. En effet, les données initiales de l'ACP informent que les valeurs maximales de la variable `NB aplFoncColorierPend` sont enregistrées par les individus E24 et L10 (respectivement 44 et 41) et que E28 enregistre une valeur maximale pour la variable `NB aplFoncTournerPend` (28 actions). Le Tableau 13 confirme les valeurs maximales de ces deux variables.

RÉSULTATS DE L'ACP COMBINÉE AVEC LA CAH. L'application de la CAH sur les coordonnées des observations dans le sous-espace vectoriel (F1,F2,F3), a retourné quatre

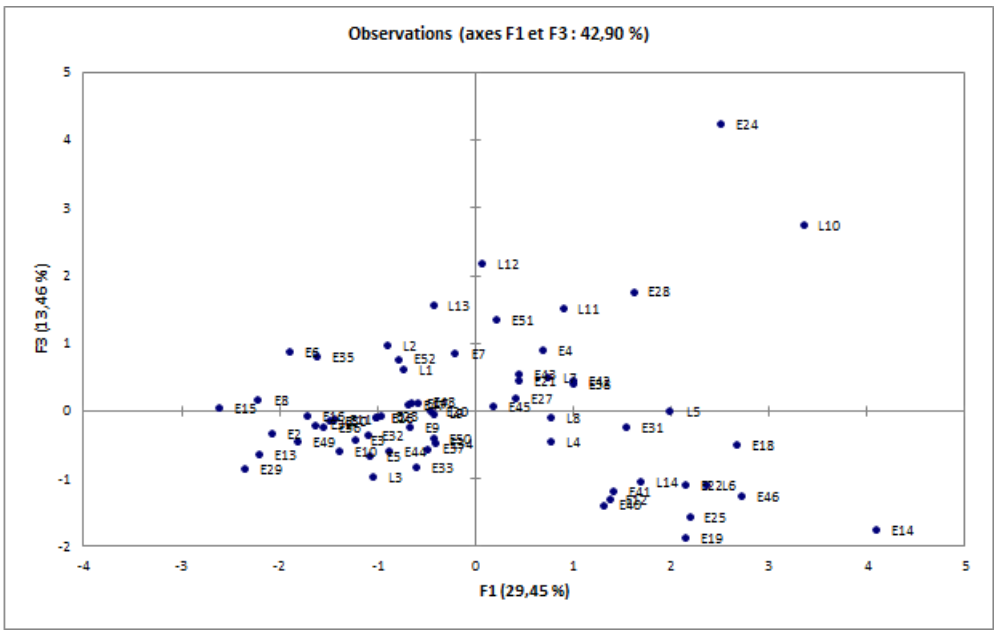


FIGURE 55 – Graphe des observations (F1,F3).

classes distinctes. En effet, la troncature du dendrogramme montre que quatre groupes homogènes se sont constitués (Figure 56) (cf. s. G.2). Le Tableau 14 donne les résultats de la CAH retournés pour chaque classe constituée.

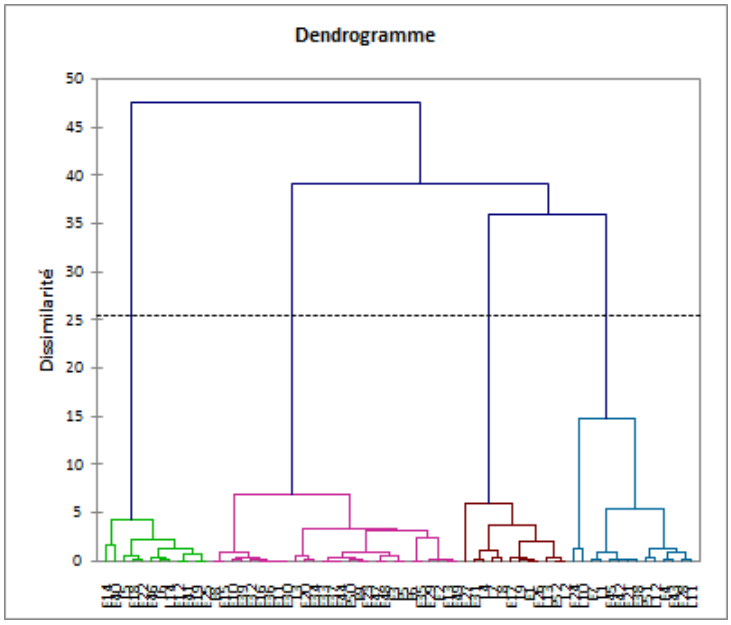


FIGURE 56 – Dendrogramme de la CAH montrant la constitution de quatre classes distinctes.

À l'issue de l'application de l'ACP une seconde fois sur les mêmes individus et sur les données initiales auxquelles est ajoutée une variable qualitative supplémentaire à savoir, *le groupe auquel appartient chaque individu*, nous avons obtenu un graphe bidimensionnel (F1,F2) des observations colorées selon le groupe auquel elles appartiennent (Figure 57). Les axes (F1,F2) ayant cumulé le maximum de variabilité, le graphe des observations

Classe	1	2	3	4
Objets	12	28	14	12
Variance intra-classe	2.09	1.25	2.95	1.67
Distance minimale au barycentre	0.45	0.50	0.66	0.22
Distance moyenne au barycentre	1.25	1.03	1.47	1.09
Distance maximale au barycentre	2.89	1.93	3.43	2.47

Tableau 14 – Résultats de la CAH montrant l'homogénéité des classes constituées.

(F1,F2) représente de manière significative la répartition de l'ensemble des participants sur 4 classes.

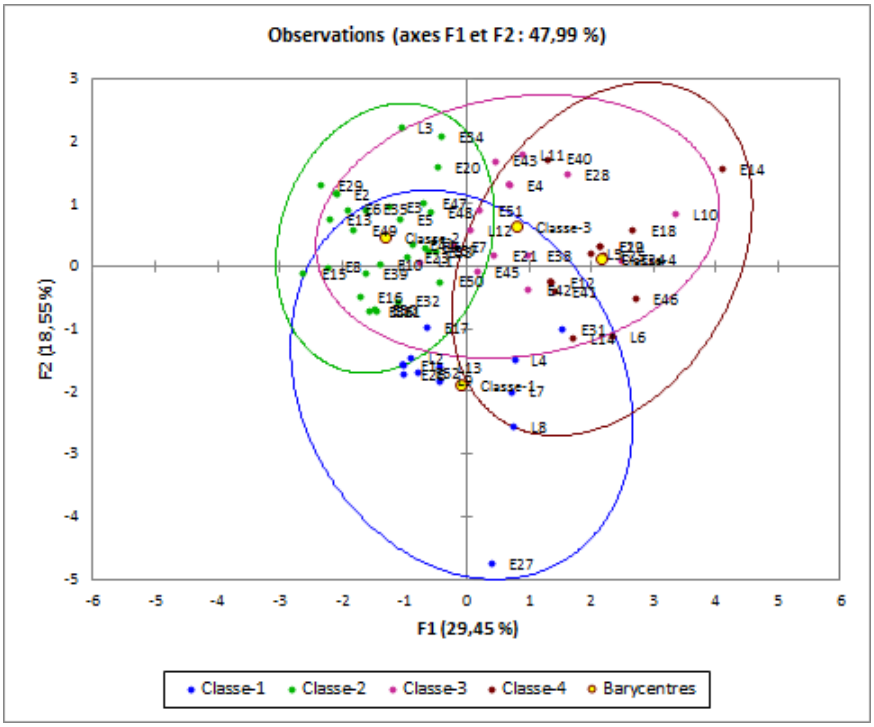


FIGURE 57 – Graphe des observations (F1,F2) montrant la répartition des individus sur 4 classes.

Le Tableau 15 complémente les résultats du graphe des observations en donnant la taille en pourcentage de chaque classe constituée, ainsi que les cordonnées des objets centraux des classes dans le sous-espace vectoriel (F1,F2,F3).

Le graphe des observations colorées montre que les objets de la *classe-1* (*les points bleus*) sont situés sur le demi-plan négatif par rapport à l'axe F2, qui est positivement corrélé avec la variable *Temps total*. Les membres de la *classe-1* représentent les participants ayant passé le moins de temps sur l'interface de PrOgO durant l'expérimentation. Ils sont peu nombreux (18.18%) et les valeurs absolues de leurs coordonnées sur l'axe F1 sont faibles. Ceci indique qu'ils ont réalisé un nombre restreint d'actions à la fois dans la scène 3D et dans l'éditeur à autocomplétion. L'objet central de cette classe possède les coordonnées (−0.42, −1.83, −0.04) dans le sous-espace vectoriel (F1,F2,F3) (Tableau 15). Cela indique que ce membre a réalisé peu d'actions liées aux axes F1 et F3 et qu'il a passé très peu de temps sur l'interface de PrOgO (−1.83 sur l'axe F2). De ce fait, la *classe-1* représente les participants ayant réalisé très peu d'actions sur l'interface de

Classe	Objets (%)	Objet central(F1)	Objet central(F2)	Objet central(F3)
1	18.18	-0.42	-1.83	-0.04
2	42.42	-0.96	0.13	-0.07
3	21.21	0.21	0.89	1.35
4	18.18	2.15	0.31	-1.08

Tableau 15 – Dimension des classes et coordonnées des objets centraux dans le sous-espace vectoriel (F1,F2,F3).

PrOgO, étant donné qu'ils ont passé très peu de temps sur son interface. Ils pourraient correspondre aux étudiants/élèves qui n'ont pas accepté de jouer et qui ont exprimé très peu d'intérêt pour l'utilisation de PrOgO durant cette expérimentation.

Les objets de la *classe-2* (*les points verts*) se situent sur le demi-plan négatif par rapport à l'axe F1 et positif par rapport à l'axe F2 (Figure 57). Cela indique que les membres de la *classe-2* ont réalisé un grand nombre d'actions dans la scène 3D et peu, voire aucune action dans l'éditeur de code. On peut confirmer cela en examinant les coordonnées de l'objet central (-0.96, 0.13, 0.07) dans le sous-espace vectoriel (F1,F2,F3) (Tableau 15). Il représente un participant qui a créé et connecté un grand nombre d'objets dans la scène 3D sans recourir à l'éditeur de code. Il a exploité le temps moyen alloué à l'expérimentation et a réalisé très peu, voire aucune action d'animation (coordonné proche de 0 sur F3). Par conséquent, la *classe-2* représente des individus qui ont passé la plupart de leur temps dans la construction des structures 3D sans recourir à l'éditeur de code et sans avoir essayé d'animer leurs réalisations. Ils sont nombreux et représentent 42.42% de l'ensemble des participants.

Les objets de la *classe-3* (*les points roses*) se situent sur le demi-plan positif par rapport aux axes F1 et F2 (Figure 57). Comparativement aux membres de la *classe-4*, leurs coordonnées sur l'axe F1 sont plus faibles. Ces objets sont également projetés près des variables définissant l'axe F3 sur le *biplot*, à savoir `NB aplFoncColorierPend et NB aplFoncTournerPend` (Figure 58). Cela indique que les membres de la *classe-3* ont exploité une bonne partie du temps alloué à la réalisation d'un grand nombre d'actions d'animation, d'autocomplétion et de sélection de code. L'objet central ayant les coordonnées (0.21, 0.89, 1.35) dans le sous-espace vectoriel (F1,F2,F3) peut confirmer cela (Tableau 15). Par conséquent, on peut déduire que la *classe-3* représente des individus se distinguant particulièrement par des actions d'animation, en plus des actions de codage et de construction. Cette classe réunit 21.21% de l'ensemble des participants.

Les objets de la *classe-4* (*les points rouges*) sont représentés dans le demi-plan positif de l'axe F1. Ils possèdent des coordonnées élevées sur l'axe F1 et des coordonnées positives sur l'axe F2 (Figure 57). Ils représentent les participants ayant réalisé le plus grand nombre d'actions de codage, tout en ayant une utilisation limitée de la scène 3D. L'objet central possède les coordonnée (2.15, 0.31, -1.08) représentant un participant qui a réalisé la plupart de ces actions dans l'éditeur de code, a exploité une bonne partie du temps alloué à l'expérimentation et a réalisé peu d'actions d'animation. La *classe-4* représente alors les participants qui ont réalisé le plus grand nombre d'actions de codage par rapport au reste des participants. Cette classe réunit 18.18% de l'ensemble des participants.

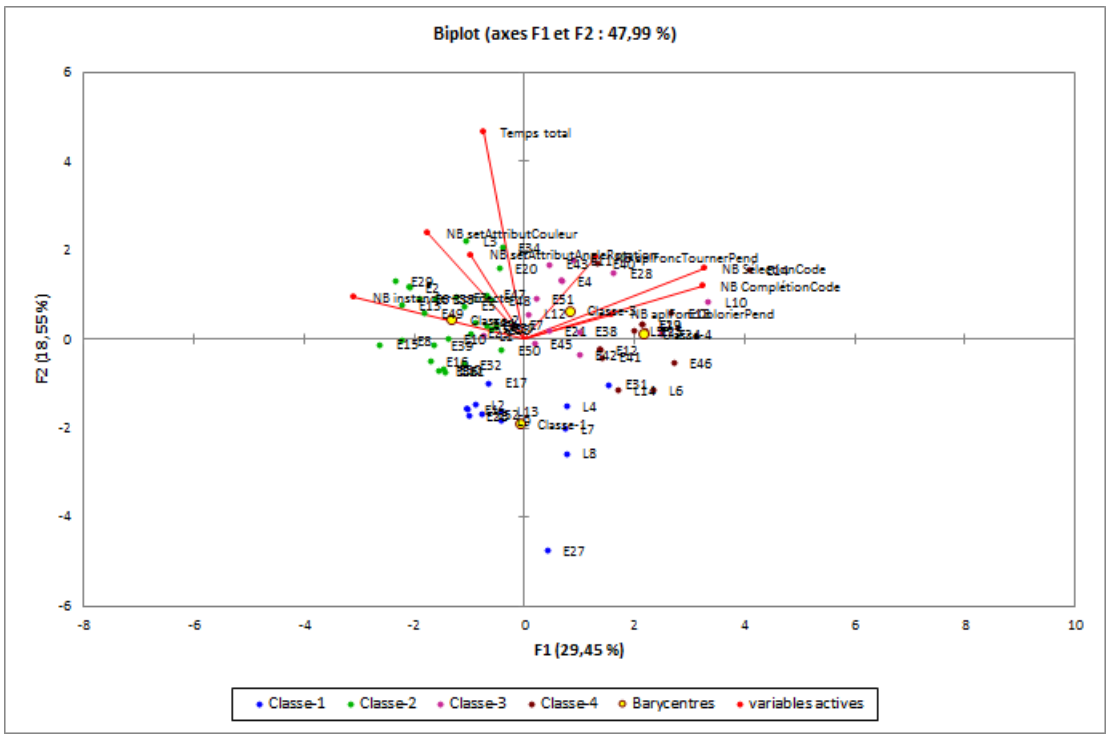


FIGURE 58 – Biplot représentant simultanément la projection des variables et des observations sur le plan (F1,F2).

6.3.3.2 *État des connaissances des étudiants/élèves*

Les résultats du test de vérification de connaissances ont montré que la distribution du score moyen global pour l'ensemble des participants suit une *loi normale* (Figure 59). Les participants ayant des scores compris entre 0.62 et 0.71 sont majoritaires, sachant que la moyenne du score moyen obtenu au test est de 0.60. La distribution normale des scores indique que les participants ayant répondu de manière *modérée* au test sont majoritaires. Les participants ayant obtenu des scores très *élevés* ou très *faibles* sont peu nombreux. En effet, les participants sont 53.73% à avoir des scores compris entre 0.45 et 0.71. Ils représentent un faible pourcentage (17.91%) à avoir des résultats plus au moins faibles compris entre 0.1 et 0.45. Enfin, ils sont 28.36% a obtenir d'excellents scores compris entre 0.71 et 0.96.

La distribution normale des scores indique également que le test est de difficulté *modérée*. En effet, l'*indice de difficulté* du test global est de 0.60 qui est aussi équivalent à la moyenne du test global (Tableau 16). Il est compris entre 0.54 et 0.65 pour chaque niveau d'apprentissage : *Reconnaître, Comprendre et Appliquer* ; ce qui est conforme aux valeurs optimales suggérées dans la littérature (cf.s.G.3.1).

Un autre résultat important est l'*indice de discrimination* qui possède une valeur comprise entre 0.45 et 0.56 pour chaque niveau d'apprentissage et 0.85 pour le test global (Tableau 16). Cela indique que le test a très bien discriminé les participants ayant obtenu des notes élevées de ceux ayant obtenu des notes faibles (cf.s. G.3.2).

L'indice de fiabilité a été estimé selon le *coefficient alpha de Cronbach* avec une valeur de 0.60 indiquant une très bonne consistance du test (cf.s. G.3.3).

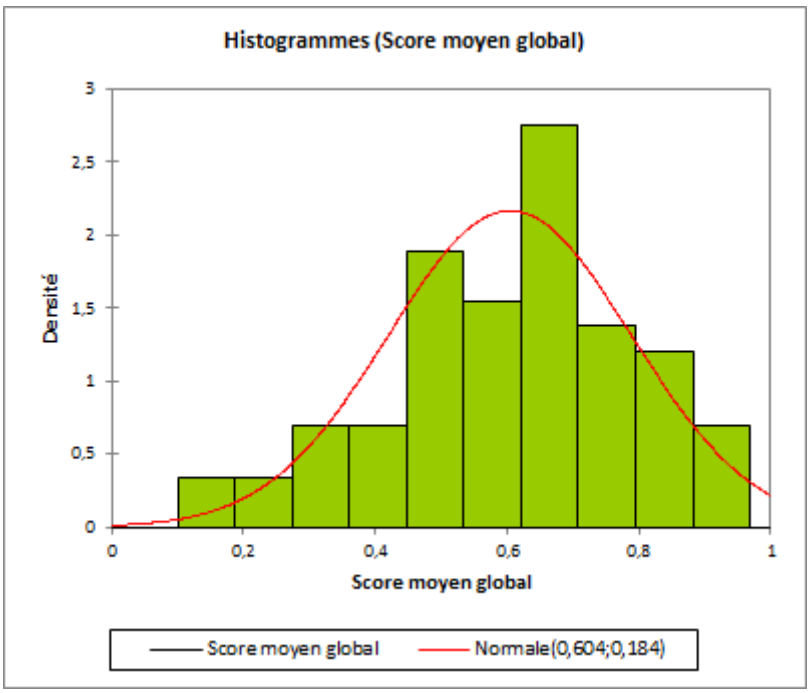


FIGURE 59 – Histogramme montrant une distribution normale du score moyen global pour l'ensemble des participants.

	A_1 [1]	A_2 [2]	A_3 [3]	A [4]
Moyenne	0.65	0.59	0.54	0.60
Écart type	0.22	0.22	0.33	0.18
Indice de difficulté (P)	0.65	0.60	0.54	0.60
Indice de discrimination (D)	0.45	0.52	0.56	0.85
Indice de fiabilité (α)			0.60	

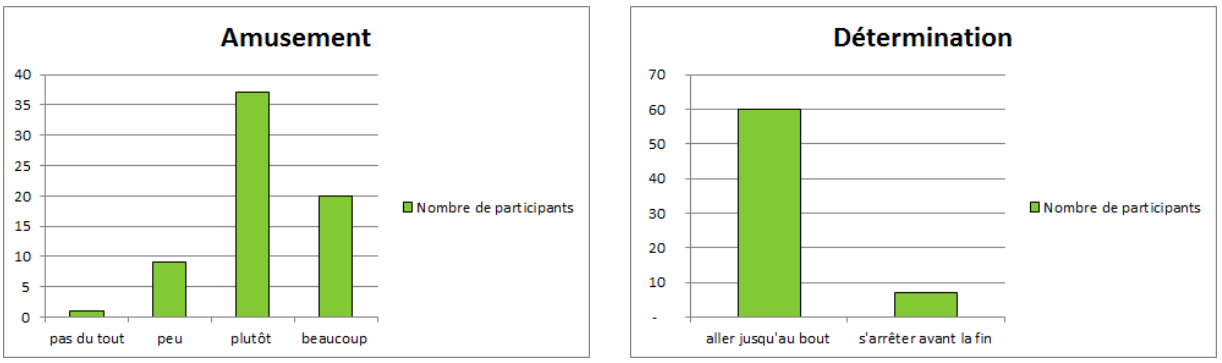
[1] Score moyen global du niveau *Reconnaître*.
[2] Score moyen global du niveau *Comprendre*.
[3] Score moyen global du niveau *Appliquer*.
[4] Score moyen du test global.

Tableau 16 – Paramètres statistiques des scores obtenus au test.

6.3.3.3 *Appréciations subjectives des étudiants/élèves*

Les participants ont exprimé leurs ressentis sur une échelle à quatre niveaux (*pas du tout, peu, plutôt, beaucoup*) concernant leur sentiment de divertissement et une échelle à deux niveaux, concernant leur sentiment de détermination (*aller jusqu'au bout, s'arrêter avant la fin*).

Les participants ont majoritairement rapporté qu'ils se sont *plutôt* ou *beaucoup* amusés (Figure 63a). Par ailleurs, quasiment tous les participants ont affirmé avoir voulu aller *jusqu'au bout* de leurs réalisations (Figure 63b). En effet certains étudiants/élèves ont clairement réussi à construire, voire animer des structures 3D significatives et complexes (Figure 61). Ceci indique que les participants étaient engagés dans le *jeu-play* et déterminés dans la réalisation de l'objectif du jeu.



(a) Sentiment d'amusement.

(b) Sentiment de détermination.

FIGURE 60 – Appréciations subjectives des participants.

6.3.4 *Discussion*

L'analyse statistique des traces d'interaction a permis d'obtenir quatre catégories distinctes d'étudiants/élèves ayant utilisé différemment PrOgO. La catégorisation et la caractérisation des participants indiquent, dans un premier temps, que les étudiants/élèves ont majoritairement accepté d'utiliser PrOgO. En effet, les participants ayant passé le moins de temps sur l'interface de PrOgO représentent un faible pourcentage (18.18%). Les participants étaient au contraire 81.82% à avoir exploité de trois différentes façons le temps qui leur a été alloué. Ils étaient 42.42% à enregistrer des actions dans la scène 3D (*classe-2*), avec pour objectif de construire des structures 3D significatives. Ils étaient 21.21% à se distinguer par le plus grand nombre d'actions d'animation, tout en enregistrant un grand nombre d'actions de codage (*classe-3*). Enfin, ils étaient 18.18% à se distinguer par le plus grand nombre d'actions d'autocompletion (*classe-4*), tout en enregistrant un faible nombre d'actions de construction et d'animation dans la scène 3D. De ce fait, les étudiants/élèves étaient au total 39.39% (*class-3 et classe-4*) à concentrer leur attention sur l'éditeur de code, en enregistrant un nombre d'actions variable, contre 42.42% à se concentrer sur leurs constructions dans la scène 3D.

Compte tenu de ces résultats, on peut confirmer l'hypothèse en lien avec la première question concernant les attitudes adoptées par les étudiants/élèves lors de l'expérimentation (cf.s. 6.3.1). En effet, les étudiants/élèves ont majoritairement exploité une grande partie du temps alloué à l'expérimentation. De plus, la *classe-3* confirme la présence d'individus ayant manipulé un grand nombre des concepts représentés dans PrOgO à la fois dans l'éditeur de code et dans la scène 3D. En effet, la *classe-3* recouvre des étudiants/élèves ayant enregistré un grand nombre d'actions à la fois dans l'éditeur à autocomplétion et dans la scène 3D. Bien que la *classe-3* ne soit pas majoritaire, ce résultat est important et ne doit pas être ignoré.

L'analyse statistique des scores obtenus au test est concluante et positive. Les résultats obtenus indiquent que la distribution des scores suit une loi normale, avec un large pourcentage d'étudiants ayant répondu au test de façon modérée. Par ailleurs, les paramètres statistiques incluant un *indice de difficulté, un indice de discrimination et un indice de fiabilité* ont été calculés avec des valeurs optimales. Cela nous amène à dire que les étudiants/élèves ont majoritairement eu de bons résultats, et que le test était de bonne

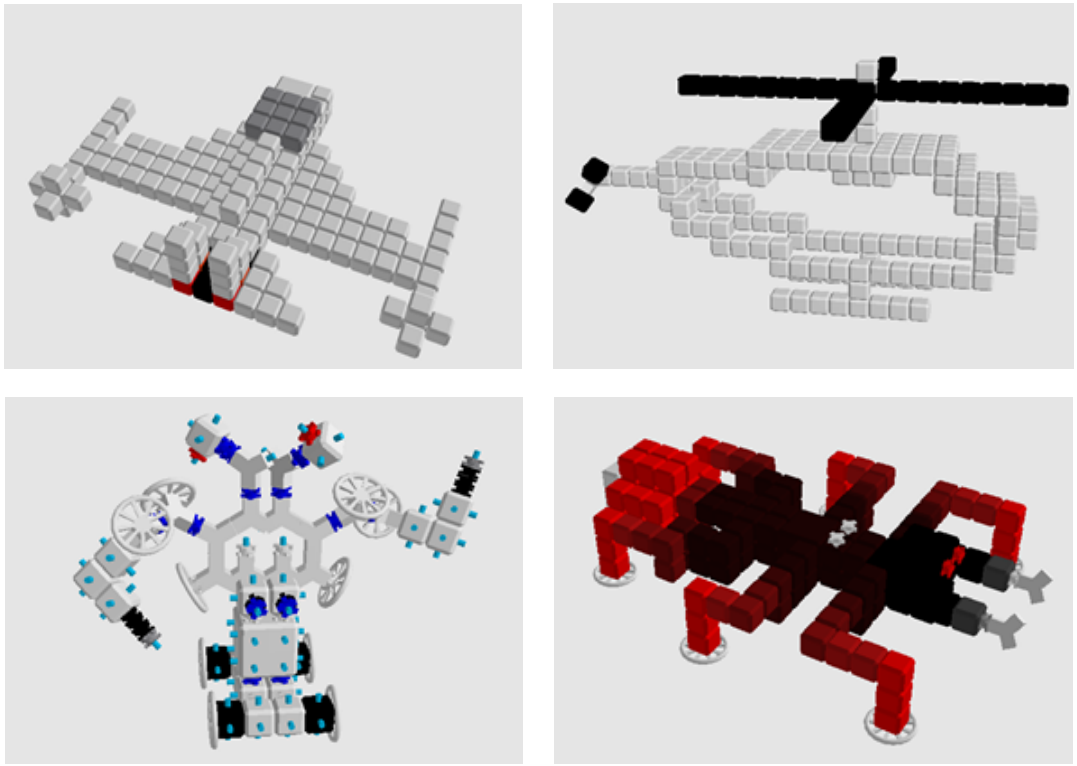


FIGURE 61 – Quelques réalisations 3D d'étudiants/élèves.

qualité. Cela indique que la majorité d'étudiants/élèves étaient capables de *reconnaître comprendre et appliquer* les concepts OO visés par le mode *Utilisation d'objets* de PrOgO en langage C++. Par conséquent, nous pouvons conclure que PrOgO est en mesure d'aider les étudiants/élèves débutants à acquérir les fondamentaux de la POO. Ce qui vient confirmer notre seconde hypothèse liée à la seconde question posée plus haut (cf.s. 6.3.1).

Notre hypothèse concernant le fait que PrOgO est amusant et engageant du point de vue des étudiants/élèves peut être confirmée. En effet, la majorité des participants ont affirmé avoir trouvé PrOgO soit *très amusant* ou *plutôt amusant*. Quasiment l'ensemble d'entre eux ont rapporté qu'ils souhaitaient achever leurs réalisations lors de l'expérimentation. De plus, certains d'entre eux ont réussi à construire, voire animer des structures 3D virtuelles significatives.

Enfin, bien que la *classe-3* ne soit pas majoritaire, les résultats d'analyse du test ont montré que la plupart des participants ont bien répondu au test qui était de difficulté modérée. Cela pourrait s'expliquer par le fait que les étudiants/élèves regardaient le code qui était généré lors de l'interaction avec les objets dans la scène 3D, sans nécessairement enregistrer des actions à l'intérieur de l'éditeur de code. De même, les étudiants/élèves pouvaient observer les résultats de leurs actions d'autocomplétion de code, de façon instantanée sur les objets 3D. Cependant, ces informations ne sont pas présentes dans les traces d'interaction, car nous n'avions pas les moyens de les mesurer.

6.3.5 *Conclusions*

À partir de ces résultats d'analyse et de leur discussion, on peut conclure que PrOgO permet la mise en place d'un jeu-play à partir duquel émerge l'apprentissage. Il est donc en mesure de réussir en tant qu'environnement d'apprentissage pour l'introduction de la POO aux débutants. En effet, les résultats obtenus ont montré, d'une part, que les étudiants/élèves ont majoritairement accepté de jouer, et certains d'entre eux ont adopté un comportement suffisamment approprié (*class-3*) leur permettant de comprendre le lien existant entre les concepts abstraits de la POO et leurs représentations visuelles à l'interface. D'autre part, l'état des connaissances des étudiants/élèves a confirmé que ces derniers étaient capables d'analyser les concepts OO abordés et de rapidement les maîtriser. De plus, les étudiants/élèves étaient engagés tout en s'amusant dans la construction et l'animation de robots ou de machines 3D, conduisant à une bonne manipulation des concepts représentés.

Nous pensons que ces résultats sont la conséquence des fondements conceptuels de PrOgO, c'est-à-dire, la métaphore de construction et d'animation utilisée pour la représentation des concepts considérés, l'illustration de cette métaphore au travers de visualisations graphiques 3D dans un système de représentation transitionnel et la situation de (*jeu-play*) qui se met en place lorsque l'étudiant/élève accepte d'interagir avec PrOgO.

Par ailleurs, nous n'avons pas pu mettre en relation l'analyse des scores obtenus au test de vérification de connaissances avec les résultats de l'ACP combinée avec la CAH, en raison de l'anonymisation des données du test. L'identification des participants n'a donc pas été possible. Dans la section suivante, nous avons tenté de mesurer la différence des scores obtenus à un pré-test et un post-test et de mettre en relation ce résultat avec les actions des étudiants dans l'interface de PrOgO. Les données du test ne sont pas anonymes, ce qui a permis d'identifier de manière claire l'ensemble des participants.

6.4 Évaluation pré/post de l'acquisition des connaissances et analyse de traces d'interaction avec ProgO v.0.7

6.4.1 *Questions de recherche et objectifs*

L'objectif de cette expérimentation est double. Il s'agit d'une part, de savoir si un apprenant débutant peut acquérir de nouvelles connaissances à l'aide de PrOgO, et d'autre part, d'identifier les actions de l'apprenant qui déterminent le gain de connaissances et donc l'apprentissage.

Nous considérons essentiellement les deux questions suivantes : *premièrement, les connaissances des apprenants débutants progressent-elles après utilisation de PrOgO ? Deuxièmement, si cette progression existe, qu'est ce qui la détermine dans les interactions des apprenants avec l'interface de PrOgO ?*

6.4.2 *Méthodologie*

Dans la réponse à la première question : les connaissances des étudiants débutants progressent-elles après utilisation de PrOgO ? Nous avons élaboré un pré-test et un post-test sur la base de la *Taxonomie de Bloom Révisée*. Nous avons également calculé des paramètres statistiques appliqués aux scores obtenus aux tests, afin de pouvoir déterminer s'ils sont significatifs.

Dans la réponse à la seconde question : qu'est ce qui détermine la progression des étudiants dans leurs interactions avec l'interface de PrOgO ? Nous avons conduit une analyse statistique des actions des participants au travers des traces d'interaction avec l'interface de PrOgO. Cette partie de la méthodologie relève du domaine des *Learning Analytics* (cf. s. 1.4.4). Afin d'arriver à identifier les actions des étudiants qui sont déterminantes dans la progression de leurs connaissances, on s'est appuyé sur l'ACP de type *pearson(n)*, appliquée à des variables d'analyse constituées à partir des traces d'interaction. Cette analyse a été réalisée sur le logiciel *XLSAT* qui s'utilise avec *Excel*.

6.4.2.1 *Contexte de l'expérimentation*

L'expérimentation a été organisée sur deux jours à la fin de l'année universitaire 2015-2016 (le 30 et 31 Mai 2016), auprès d'étudiants de première année universitaire en *DUT Métiers du Multimédia et de l'Internet à l'IUT du Puy en Velay* (Université d'Auvergne).

Au cours de leur première année, les étudiants suivent un module d'*Algorithmique et de programmation* dans lequel ils sont initiés à l'algorithmique et à la programmation Web avec le langage PHP : *types de données, variables, opérateurs, structures de contrôle conditionnelles et itératives, tableaux, procédures et fonctions*. Les étudiants n'ont pas de connaissances évoluées en POO et en langage C++. Le nombre de participants était de 51 le premier jour et de 48 le deuxième jour.

Chacune des deux séances de l'expérimentation a duré 1h45mn. La première a été consacrée à l'*utilisation d'objets* et la seconde à la *création de classes*. Lors de chaque séance, les participants ont répondu à un pré-test d'évaluation de connaissances pendant 20mn, puis pendant 1h, ils ont manipulé PrOgO v.0.7 en ayant à disposition un tutoriel d'utilisation décrivant son interface (cf.s. 5.2.2). Après cela, les participants ont répondu à un post-test d'évaluation de connaissances pendant 20mn. Les tests de vérification de connaissances était accompagnés d'exemples de programmes en langages C++ (cf. s. F.2). Enfin, pendant 5mn supplémentaires, ils ont été amenés à exprimer leurs opinions et sentiments de divertissement et de détermination dans la réalisation de l'objectif du *jeu-play* (cf. s. F.3).

Lors de la séance d'utilisation d'objets, les participants étaient amenés à donner libre court à leur imagination pour construire et animer des robots virtuels ou des structures mécaniques 3D. Lors de la seconde séance consacrée à la création de classes, les participants ont été amenés à créer de nouvelles classes à partir de leurs constructions réalisées lors la séance précédente. Ils ont été amenés à améliorer leurs réalisations en modifiant leurs apparences (modifier les anciennes données membres ou ajouter de nouvelles), et en modifiant leurs comportements (modifier leurs animations).

Durant cette expérimentation, les traces d'interaction des participants avec l'interface de PrOgO ont été sauvegardées dans des fichiers externes (*.csv).

6.4.2.2 *Collecte et analyse de données*

ÉLABORATION DES PRÉ-TESTS ET POST-TESTS ET ANALYSE DES SCORES. La procédure d'élaboration des pré-tests et post-tests pour chacune des deux séances de cette expérimentation, est semblable à celle de la précédente expérimentation (cf.s. 6.3). Le contenu de chaque test a été élaboré suivant les trois niveaux inférieurs de la taxonomie de Bloom révisée des objectifs d'apprentissage (Annexe E).

Les objectifs d'apprentissage visés au cours de la séance *utilisation d'objets* sont identiques à ceux de la précédente expérimentation (cf.s. 6.3). Nous avions alors réutilisé le même test pour constituer le contenu du pré-test et du post-test de cette séance expérimentale (Tableau 10).

Les objectifs d'apprentissage visés lors de la seconde séance expérimentale (*Création de classes*) concernent des concepts de base essentiels à la compréhension de la classe et de son rôle dans un programme OO. Les étudiants sont amenés à *créer des classes* (cf.s. 5.2.3.2), en vue d'apprendre les concepts suivants :

- *Le rôle d'une classe* : créer une classe revient à créer une *nouvelle abstraction*, ou un *nouveau type de données*.
- *Caractéristiques d'une classe* : une classe possède des *données et fonctions membres*.
- *Encapsulation dans une classe* : une classe *encapsule* ses données et fonctions membres (règles d'encapsulation avec les mots clé `public` et `private`).
- *Constructeur de classe* : une classe possède un constructeur qui permet d'initialiser ses instances (objets) à leur création.
- *Utilisation d'une classe* : inclusion du fichier de déclaration de la classe, insatanciation de la classe et invocation de méthodes sur ses instances.

PrOgO v.0.7 est conçu de façon à introduire l'ensemble de ces concepts et d'amener l'étudiant à les implémenter en langage de programmation C++. Par conséquent, le pré-test et post-test doivent estimer en terme de scores, la capacité des étudiants à :

- *identifier et nommer* ces différents concepts,
- *expliquer* certains éléments en lien avec ces concepts,
- *implémenter* ces concepts en langage de programmation C++.

Les pré-tests et post-tests de la séance *Création de classes* sont constitués conformément aux trois premiers niveaux hiérarchiques de la taxonomie de Bloom révisée : *Reconnaître, Comprendre et Appliquer*. Chacun des deux tests est constitué de 13 questions (cf. s. F.2) : 3 questions à réponses courtes, 5 questions à réponses longues, 3 question à choix unique, 2 questions à choix multiples. Le Tableau 17 résume les objectifs des différentes questions constituant les deux tests.

De la même manière que la précédente expérimentation (cf.s. 6.3), trois paramètres statistiques sont calculés (*indice de difficulté*, *indice de discrimination*, *indice de fiabilité*) (cf.s. G.3), afin de comparer de manière significative les scores obtenus au pré-test et au post-test.

COLLECTE ET ANALYSE DE TRACES NUMÉRIQUES D'INTERACTION. Le modèle de traces d'interaction de PrOgO v.0.7 est une extension du modèle de PrOgO v.0.6 (Annexe D). Ces traces comprennent les actions liées à la création d'objets dans la scène 3D, la modification de valeurs d'attributs d'objets et les appels de méthodes sur les objets. Les traces supplémentaires générées lors du premier mode *Utilisation d'objets*,

Niveau de la taxonomie	Question
Niveau 1 : *Reconnaître*	- Identifier une nouvelle classe par son nom dans un programme édité dans PrOgO
	- Reconnaître une déclaration de classe, une définition de classe et une utilisation de classe dans des programmes édités dans PrOgO
	- Listez les mots clés permettant d'implémenter l'encapsulation dans une classe
	- Identifier le constructeur d'une classe
Niveau 2 : *Comprendre*	- Conclure sur le rôle d'une classe
	- Conclure sur les données et fonctions membres d'une classe
	- Conclure sur la différence entre une déclaration de classe et une définition de classe
	- Conclure sur le rôle de l'encapsulation dans une classe
	- Conclure sur le rôle du constructeur dans une classe et sur ses propriétés
	- Décrire les étapes d'utilisation d'une nouvelle classe
Niveau 3 : *Appliquer*	- Implémenter une déclaration de classe
	- Implémenter une définition de classe
	- Implémenter une utilisation de classe

Tableau 17 – Description des questions constituant le pré-test et le post-test utilisés lors de la séance *Création de classes*.

comprennent essentiellement *les actions de modification de noms d'objets et de suppression de lignes de code dans l'éditeur à auto-complétion, les actions d'exécution pas à pas de code et les actions de sélection d'items dans la hiérarchie d'objets*. Le second mode *Création de classes* permet de générer d'autres actions telles que *créer une nouvelle classe, instancier une nouvelle classe créée, passer au mode 2 et revenir au mode 1*.

La procédure d'analyse des traces numériques d'interaction, que nous avons suivie au cours de cette expérimentation est semblable à celle de la précédente expérimentation (cf.s. 6.3). Les traces brutes ont été traitées afin de constituer des variables d'analyse de l'ACP (cf. s. G.1).

À l'issue de la séance *Utilisation d'objets*, huit variables d'analyse sont constituées à partir des traces collectées. Ces variables donnent pour chaque participant, le nombre d'actions réalisées sur l'interface de PrOgO et le temps total passé sur cette interface durant l'expérimentation (Tableau 18). Les actions réalisées dans la scène 3D sont quantifiées et exprimées en *nombre d'actions d'instanciation de classes et de connexion d'objets, nombre d'actions de modification de valeurs d'attributs et nombre d'appels de méthodes*. Les actions réalisées dans l'éditeur de code sont quantifiées et exprimées en *nombre d'actions de complétion de code et en nombre d'autres actions de code*. Les actions *d'exécution pas à pas de code* et de *sélection d'items dans la hiérarchie d'objets* sont également quantifiées.

À ces huit variables (dites *variables actives*), est ajoutée la différence des scores obtenus au pré-test et post-test, comme *variable quantitative supplémentaire* (Tableau 18).

À l'issue de la séance *Création de classes*, 12 variables d'analyse sont constituées à partir des traces collectées, dont 8 sont identiques aux 8 variables constituées lors de la séance *Utilisation d'objets* et 4 en relation avec des actions possibles uniquement dans le mode 2 (Tableau 18). Étant donné que PrOgO v.0.7 autorise le retour au mode 1, il est important de constituer des variables en lien avec le premier mode *Utilisation d'objets*, d'où les 8 premières variables.

L'ensemble des participants désignés par leurs identifiants constituent *les observations* de l'ACP. L'objectif étant d'identifier et de visualiser les *corrélations* existantes entre la variable supplémentaire `progression` avec les variables d'analyse actives constituées lors de chaque séance.

Variable	Signification	séance 1	séance 2
NB Instancier-Connecter	Nombre d'actions d'instanciation de classes et de connexion d'objets dans la scène 3D	oui	oui
NB modifAttributs	Nombre d'actions de modification de valeurs d'attributs dans la scène 3D `couleur` et `angleDeRotation`	oui	oui
NB appelsFonctions	Nombre d'appels des méthodes `colorierPendant()` et `tournerPendant()` dans la scène 3D	oui	oui
NB complétionCode	Nombre d'actions de complétion de code	oui	oui
NB autresCodeActions	Nombre d'autres actions réalisées dans l'éditeur de code (sélection de code, modification de noms d'objets et suppression de lignes de code)	oui	oui
NB executerCode pas à pas	Nombre d'actions d'exécution pas à pas de code	oui	oui
NB sélection itemsHiérarchiques	Nombre d'actions de sélection d'items dans la hiérarchie d'objets	oui	oui
temps total	Temps total passé sur l'interface de PrOgO durant l'expérimentation	oui	oui
NB créer Classes	Nombre d'actions de création de classes modélisant des constructions réalisées au cours du mode *Utilisation d'objets*	non	oui
NB instancier Classes	Nombre d'actions d'instanciation d'une nouvelle classe créée	non	oui
NB passages Mode 2	Nombre d'actions marquant le passage au mode *Création de classes*	non	oui
NB retours Mode 1	Nombre d'actions marquant le retour au mode *Utilisation d'objets*	non	oui
progression	Variable quantitative supplémentaire indiquant la différence des scores entre le pré-test et le post-test de chaque séance	oui	oui

Tableau 18 – Variables d'analyse de l'ACP constituées à partir des données issues des deux séances *Utilisation d'objets* et *Création de classes*.

6.4.3 *Résultats*

6.4.3.1 *Évaluation pré/post de l'acquisition des connaissances*

UTILISATION D'OBJETS. Les résultats obtenus au pré-test et au post-test de la séance *Utilisation d'objets*, indiquent que les scores obtenus au post-test sont en progression par rapport aux scores obtenus au pré-test. Pour chacun des trois niveaux *Reconnaître, Comprendre et Appliquer* et pour le test dans son ensemble, la moyenne des scores des participants a significativement augmenté dans le post-test (Tableau 19). Les résultats montrent que les participants ont principalement progressé dans les questions des niveaux *Comprendre et Appliquer*. Les moyennes des scores des deux niveaux obtenus aux deux tests sont respectivement (0.34/0.50 et 0.54/0.76).

L'indice de difficulté P (équivalent à la moyenne) pour chacun des trois niveaux et pour l'ensemble du test, possède une valeur rentrant dans la plage optimale (entre 0.3 et 0.8) (cf.s.G.3.1). Cela indique que la difficulté des tests était modérée. L'indice P a également augmenté dans le post-test. Ses valeurs respectives pour chacun des trois niveaux sont (0.40/0.48, 0.34/0.50 et 0.54/0.76). Ses valeurs pour l'ensemble du test sont (0.40/0.54). Cela indique que le post-test a été perçu moins difficile que le pré-test.

Le pré-test et le post-test possèdent le même indice de discrimination D (0.42). Cela indique que les deux tests ont discriminé de la même façon les participants ayant obtenu des notes élevées de ceux ayant obtenu des notes faibles. L'indice D étant supérieur à 39, les deux tests ont très bien discriminé l'ensemble des participants.

Au niveau *Appliquer*, l'indice D de chacun des deux tests, est très grand comparativement aux autres niveaux. Cela indique un grand écart de scores entre les participants ayant répondu correctement aux exercices d'implémentation de ceux n'ayant pas répondu correctement.

Enfin, l'indice de fiabilité α du pré-test possède une valeur inférieure à celle souhaitée (inférieur à 0.6) contrairement à celui du post-test (0.63) (cf.s. G.3.3). Étant donné que le pré-test est identique au post-test, l'indice α montre que le test devient fiable et consistant quand les étudiants gagnent des connaissances.

	pré-test				post-test			
	A_1	A_2	A_3	A	A_1	A_2	A_3	A
Moyenne	0.40	0.34	0.54	0.40	0.48	0.50	0.76	0.54
Écart type	0.25	0.16	0.36	0.17	0.27	0.22	0.34	0.20
Indice de difficulté (P)	0.40	0.34	0.54	0.40	0.48	0.50	0.75	0.54
Indice de discrimination (D)	0.62	0.38	0.94	0.42	0.68	0.54	0.77	0.42
Indice de fiabilité (α)	0.50				0.63			

A_i est le score moyen global du niveau i.

i vaut 1 pour *Reconnaître*, 2 pour *Comprendre* ou 3 pour *Appliquer*.

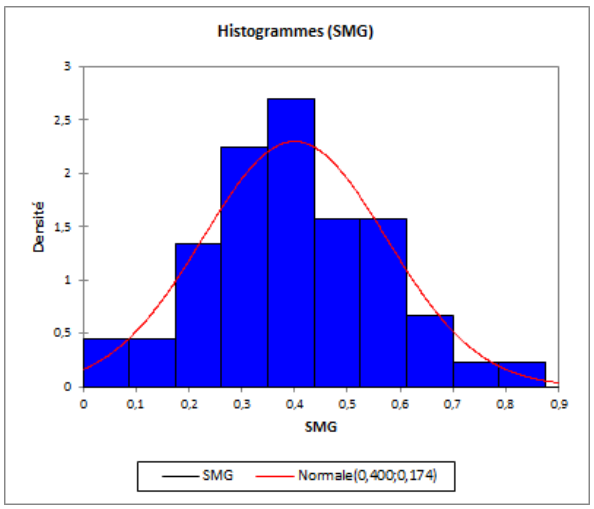
A est le score moyen du test global.

Tableau 19 – Paramètres statistiques des scores obtenus au pré-test et au post-test de la séance *Utilisation d'objets*.

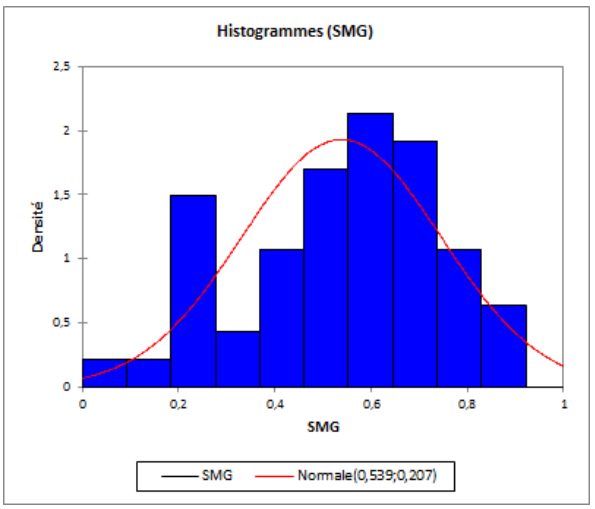
On peut voir également la progression des scores des participants en observant les histogrammes du score moyen global du pré-test et du post-test (Figure 62). Les histogrammes indiquent que la fréquence du score moyen global supérieur à la moyenne est plus dense sur le post-test comparativement au pré-test, tandis que la fréquence du score moyen global inférieur à la moyenne est plus dense sur le pré-test comparativement au post-test.

CRÉATION DE CLASSES. Les résultats obtenus aux tests de la séance *Création de classes*, indiquent que les scores obtenus au post-test sont en bonne progression par rapport aux scores obtenus au pré-test. Pour chacun des trois niveaux *Reconnaître, Comprendre et Appliquer* et pour l'ensemble du test, la moyenne des scores des participants a augmenté de manière significative dans le post-test (Tableau 20). Les participants ont progressé dans la réponse aux questions des trois niveaux. Les moyennes des scores des trois niveaux et du test global obtenus aux deux tests sont respectivement (0.47/0.61, 0.36/0.57, 0.01/0.13 et 0.32/0.50).

L'indice de difficulté P de chacun des niveaux *Reconnaître et Comprendre* possède une valeur rentrant dans la plage optimale (entre 0.3 et 0.8) (cf.s.G.3.1). Sa valeur respective pour chacun des deux niveaux est de (0.47 et 0.36) pour le pré-test et de (0.61 et 0.57) pour le post-test. En revanche le niveau *Appliquer* a été perçu très difficile par les participants. Sa valeur est de 0.01 au pré-test et 0.12 au post-test.



(a) Score moyen global (SMG) du pré-test.



(b) Score moyen global (SMG) du post-test.

FIGURE 62 – Histogrammes du score moyen global de l'ensemble des participants obtenu au pré-test et au post-test lors de la séance *Utilisation d'objets*.

Globalement, l'indice de difficulté P (équivalent à la moyenne) pour chacun des trois niveaux et pour l'ensemble du test a augmenté dans le post-test. Cela indique que le post-test a été perçu moins difficile que le pré-test.

L'*indice de discrimination* D de chacun des pré-test et post-test est très faible (respectivement 0.09 et 0.18). Cela indique que les deux tests ont discriminé de manière insuffisante les participants ayant obtenu des notes élevées de ceux ayant obtenu des notes faibles. L'indice D des niveaux *Reconnaître et Comprendre* de chacun des deux tests est supérieur à 0.39. En revanche l'indice D du niveau *Appliquer* est très faible pour le pré-test et suffisant pour le post-test (respectivement 0.18 et 0.46). Cela indique que les questions du niveau *Appliquer* ont bien discriminé les participants.

Enfin, l'indice de fiabilité α de chacun du pré-test et du post-test possède une valeur inférieure à celle souhaitée, respectivement (0.43 et 0.40) (inférieur à 0.6) (cf.s. G.3.3).

Bien que la moyenne des scores de l'ensemble des participants a progressé dans le post-test, l'indice α montre que le test n'est pas suffisamment consistant.

	pré-test				post-test			
	A_1	A_2	A_3	A	A_1	A_2	A_3	A
Moyenne	0.47	0.36	0.01	0.32	0.61	0.57	0.13	0.50
Écart type	0.19	0.17	0.05	0.12	0.22	0.19	0.23	0.14
Indice de difficulté (P)	0.47	0.36	0.01	0.32	0.61	0.57	0.12	0.49
Indice de discrimination (D)	0.47	0.46	0.18	0.09	0.52	0.41	0.46	0.18
Indice de fiabilité (α)	0.43				0.40			

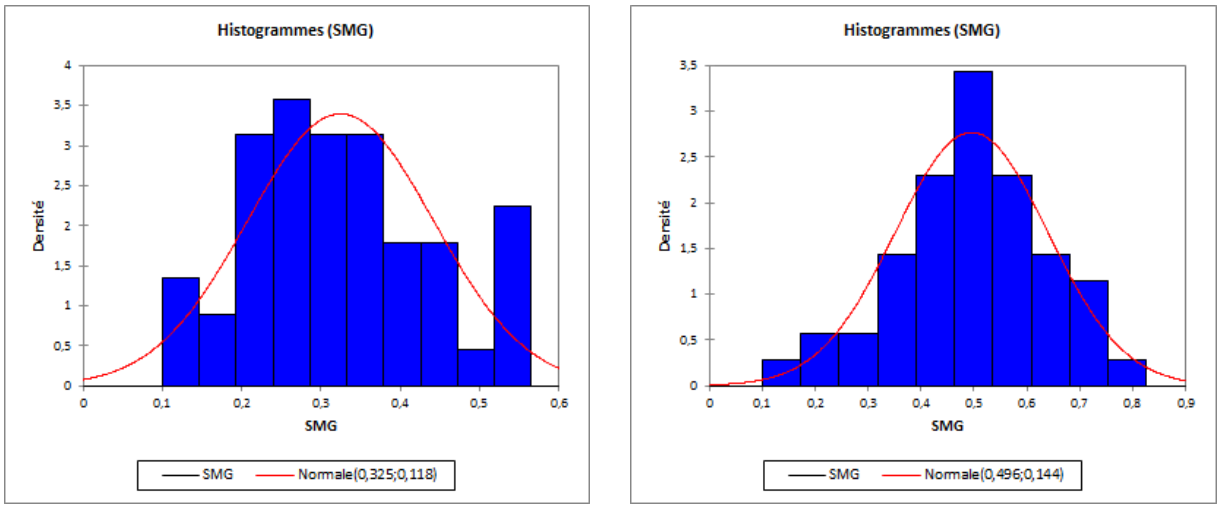
A_i est le score moyen global du niveau i.

i vaut 1 pour *Reconnaître*, 2 pour *Comprendre* ou 3 pour *Appliquer*.

A est le score moyen du test global.

Tableau 20 – Paramètres statistiques des scores obtenus au pré-test et au post-test de la séance *Création de classes*.

On peut observer sur les histogrammes du score moyen global du pré-test et du post-test, que la distribution des scores devient plus homogène sur le post-test (Figure 63). La distribution du score moyen global pour l'ensemble des participants au post-test suit une *loi normale*. Les participants ayant des scores s'approchant de la moyenne (0.50) sont majoritaires, tandis que les participants ayant obtenu des scores très *élevés* ou très *faibles* sont peu nombreux.



(a) Score moyen global (SMG) du pré-test.

(b) Score moyen global (SMG) du post-test.

FIGURE 63 – Histogrammes du score moyen global de l'ensemble des participants obtenu au pré-test et au post-test lors de la séance *Création de classes*.

6.4.3.2 *Analyse de traces d'interaction*

UTILISATION D'OBJETS. L'ACP a retourné huit facteurs (composantes principales ou axes) calculés à partir des variables d'analyse issues de la séance *Utilisation d'objets* (Tableau 18). Les *valeurs propres* des trois premiers facteurs ont permis de cumuler 66.89% de la *variabilité* initiale des données (Tableau 21). Nous avons retenu les trois premiers facteurs et ignoré les cinq autres. Il convient de retenir un nombre restreint de facteurs ayant cumulé un maximum de variabilité (cf. s. G.1.2).

	F1	**F2**	**F3**	F4	F5	F6	F7	F8
Valeur propre	**2.60**	**1.70**	**1.05**	0.88	0.72	0.59	0.44	0.02
Variabilité (%)	**32.57**	**21.22**	**13.11**	10.59	9.02	7.40	5.54	0.20
% cumulé	**32.57**	**53.79**	**66.89**	77.84	86.86	94.26	99.78	100

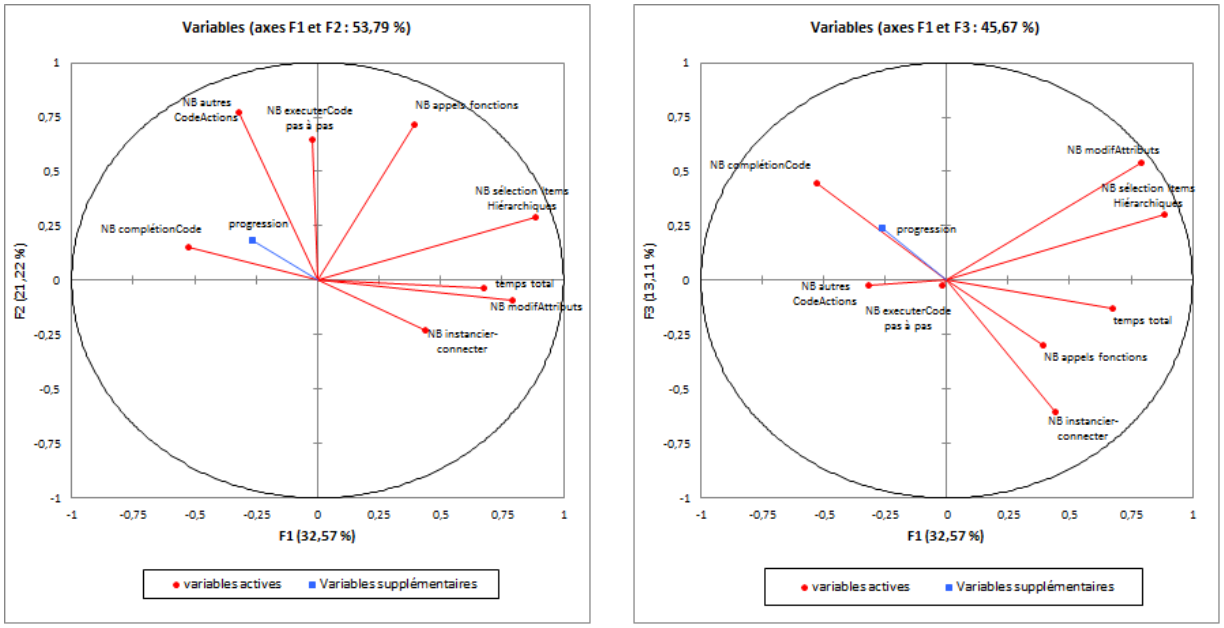
En gras sont les facteurs retenus avec leurs valeurs propres, variabilités et variabilités cumulées exprimées en (%).

Tableau 21 – Composantes principales retournées par l'ACP sur les variables d'analyse constituées suite à la séance *Utilisation d'objets*.

Les cercles de corrélation (F1,F2) et (F1,F3) montrent que la variable `progression` n'est pas bien représentée sur les axes F1, F2 et F3 (Figure 64). En effet, la variable `progression` se retrouve proche du centre de chaque cercle. Le Tableau 22 confirme que

la variable *progression* est faiblement corrélée avec les axes F1, F2 et F3. Sa corrélation avec ces trois axes est respectivement (-0.26, 0.18 et 0.24).

Les variables les plus fortement corrélées avec l'axe F1 sont `NB modifAttributs`, `NB sélection itemsHiérarchiques` et `temps total`. Leurs valeurs de corrélation avec l'axe F1, sont respectivement 0.79, 0.89 et $0,68$. L'axe F2 est positivement corrélé avec les variables `NB appelsFonctions`, `NB autresCodeActions` et `NB executerCode pas à pas`. Les valeurs de corrélation respectives sont 0.72 , 0.77 et 0.65. Enfin, la variable définissant l'axe F3 est `NB Instancier-Connecter`, avec une valeur de corrélation de -0.60.



(a) Cercle des corrélations (F1,F2). (b) Cercle des corrélations (F1,F3).

FIGURE 64 – Cercles des corrélations montrant la corrélation de la variable `progression` avec les variables actives sur les axes F1 - F2 et F1 - F3.

Variables	F1	F2	F3
NB Instancier-Connecter	0.44	-0.23	**-0.60**
NB modifAttributs	**0.79**	-0.09	0.54
NB appelsFonctions	0.39	**0.72**	-0.30
NB complétionCode	-0.53	0.15	0.44
NB autresCodeActions	-0.32	**0.77**	-0.02
NB executerCode pas à pas	-0.02	**0.65**	-0.02
NB sélection itemsHiérarchiques	**0.89**	0.29	0.30
Temps total	**0.68**	-0.03	-0.13
progression	-0.26	0.18	0.24

Les valeurs en gras correspondent aux corrélations les plus fortes entre les variables et les facteurs.

Tableau 22 – Corrélations des variables d'analyse avec les axes F1, F2 et F3.

La matrice de corrélation retournée par l'ACP, donne les corrélations entre chaque deux variables, à la fois actives et supplémentaires quantitatives. Le Tableau 23 donne

un extrait de la matrice de corrélation et montre la corrélation de la variable supplémentaire `progression` avec le reste des variables actives. On peut observer que la variable `progression` est bien corrélée avec la variable `NB autresCodeActions` comparativement aux reste des variables. Cela indique que les étudiants ayant amélioré leurs scores dans le post-test, sont ceux qui ont enregistré des actions dans l'éditeur de code. Ces actions sont différentes de l'action d'autocomplétion de code, et comprennent les actions de sélection de code, modification de noms d'objets et suppression de lignes de code (Tableau 18).

Variables actives	progression
NB Instancier-Connecter	-0.25
NB modifAttributs	-0.03
NB appelsFonctions	-0.06
NB complétionCode	0.24
NB autresCodeActions	0.37
NB executerCode pas à pas	-0.003
NB sélection itemsHiérarchiques	-0.01
Temps total	-0.26

Tableau 23 – Corrélation (*pearson(n)*) de la variable `progression` avec le reste des variables actives.

Les trois premiers facteurs (F1, F2 et F3), sont représentatifs des actions décrivant et caractérisant le mieux l'ensemble des individus. La non corrélation de la variable `progression` avec ces trois facteurs, signifie que les individus ont majoritairement réalisé des actions différentes de la variable *NB autresCodeActions*. En effet, le Tableau 24 confirme que les individus ont enregistré en moyenne 9.04 actions de codage différentes de l'autocomplétion (*NB autresCodeActions*), et au maximum 27 actions. Ce qui est un nombre faible comparativement aux autres actions enregistrées.

Variables	Minimum	Maximum	Moyenne
NB Instancier-Connecter	17	822	254
NB modifAttributs	0	173	44.59
NB appelsFonctions	0	106	19.55
NB complétionCode	0	165	11.40
NB autresCodeActions	0	27	9.04
NB executerCode pas à pas	0	4	0.82
NB sélection itemsHiérarchiques	2	208	84.94
Temps total	0.02	0.05	0.04
progression	-0.15	0.51	0.14

Tableau 24 – Statistiques descriptives de l'ACP pour l'ensemble des participants.

CRÉATION DE CLASSES. L'ACP a retourné 12 facteurs calculés à partir des 12 variables d'analyse issues de la séance *Création de classes* (Tableau 18). Les *valeurs propres* des trois premiers facteurs ont permis de cumuler 60.65% de la *variabilité* initiale des données (Tableau 25). Nous avons retenu les trois premiers facteurs, car il convient de retenir un nombre restreint de facteurs ayant cumulé un maximum de variabilité (cf. s. G.1.2).

	F1	**F2**	**F3**	F4	F5	F6	F7	F8	F9	F10	F11	F12
Valeur propre	**3.55**	**2.23**	**1.5**	1.18	0.94	0.77	0.65	0.51	0.40	0.22	0.04	0.02
Variabilité (%)	**29.60**	**18.55**	**12.50**	9.82	7.80	6.41	5.44	4.23	3.35	1.86	0.30	0.13
% cumulé	**29,60**	**48,15**	**60,65**	70,47	78,27	84,68	90,12	94,35	97,70	99,57	99,86	100

En gras sont les facteurs retenus avec leurs valeurs propres, variabilités et variabilités cumulées exprimées en (%).

Tableau 25 – Composantes principales retournées par l'ACP sur les variables d'analyse constituées suite à la séance *Création de classes*.

Le Tableau 26 donne un extrait de la matrice des corrélations et montre la corrélation de la variable `progression` avec les autres variables actives ayant servi à la constitution des axes (F1, F2 et F3). On peut observer qu'il n'existe pas de fortes corrélations de la variable `progression` avec les autres variables. En effet, les corrélations *Pearson(n)* possèdent des valeurs inférieurs à 0.5.

On peut observer que les variables avec lesquelles les corrélations sont maximales sont *NB modifAttributs, NB sélection itemsHiérarchiques, NB appelsFonctions, NB autresCodeActions, NB passages Mode 2 et NB retours Mode 1*. Cela peut être confirmé par le cercle des corrélations (F1,F2) qui montre la projection de la variable `progression` dans le même sens que les variables énumérées ici (Figure 65). Cela donne peu d'indications sur les actions qui déterminent la différence des scores obtenus au post-test et au pré-test. Les valeurs des corrélations de la variable `progression` montrent que les actions les plus déterminantes sont les actions de sélection d'items dans la hiérarchie d'objets et les actions de modification de valeurs d'attributs dans la scène 3D, les valeurs de corrélation de *progression* avec *NB modifAttributs et NB sélection itemsHiérarchiques* étant respectivement 0.43 et 0.51.

Variables actives	progression
NB Instancier-Connecter	0,16
NB modifAttributs	0,43
NB appelsFonctions	0,21
NB complétionCode	0,17
NB autresCodeActions	0,21
NB executerCode pas à pas	0,19
NB sélection itemsHiérarchiques	0,51
Temps total	0,19
NB créer Classes	0,16
NB instancier Classes	0,01
NB passages Mode 2	0,26
NB retours Mode 1	0,26

Tableau 26 – Corrélation (*pearson(n)*) de la variable `progression` avec le reste des variables actives.

6.4.3.3 *Appréciations subjectives*

Les participants ont exprimé leurs ressentis sur une échelle à quatre niveaux (*pas du tout, peu, plutôt, beaucoup*) concernant leur sentiment de divertissement et une échelle à

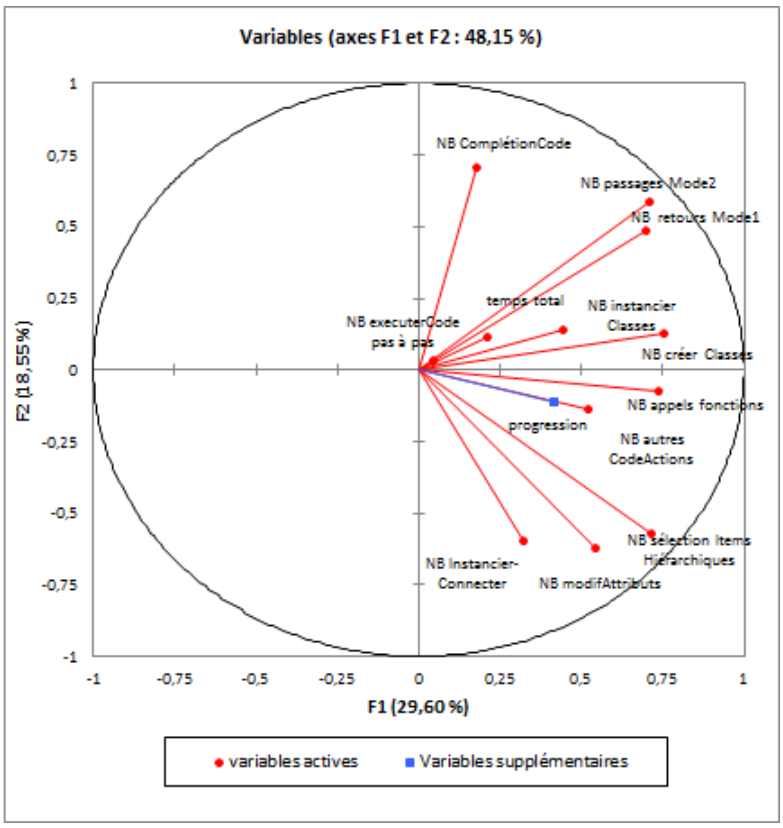


FIGURE 65 – Cercle des corrélations (F1,F2) montrant la dépendance de la variable `progression` avec les variables actives.

deux niveaux, concernant leur sentiment de détermination (*aller jusqu'au bout*, *s'arrêter avant la fin*).

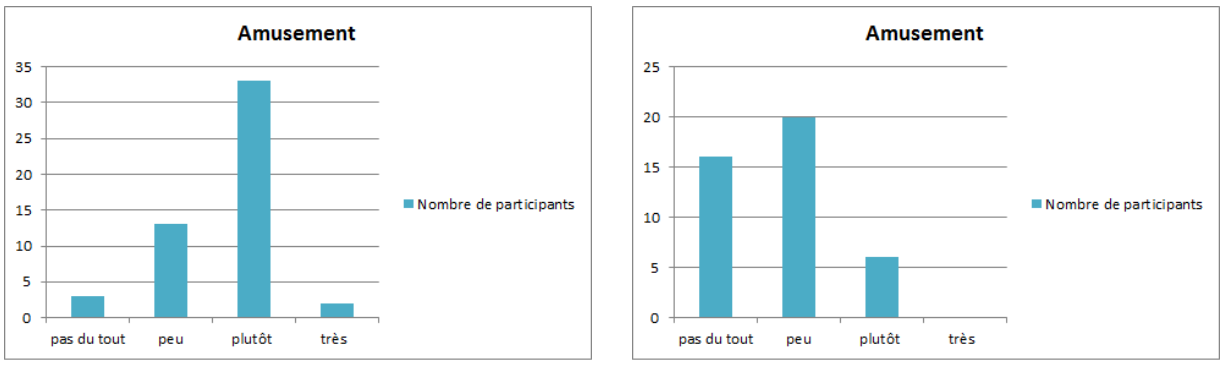
Les participants ont rapporté avoir trouvé la séance *Utilisation d'objets* plus amusante que la séance *Création de classes*. En effet, lors de la séance *Utilisation d'objets*, ils ont majoritairement affirmé qu'ils se sont *plutôt* ou *peu* amusés (Figure 66a), et lors de la séance *Création de classes*, ils ont majoritairement rapporté qu'ils se sont *peu* ou *pas du tout* amusés (Figure 66b).

Les participants manquaient d'engagement et de détermination lors de l'expérimentation. Lors de la séance *Utilisation d'objets*, ils étaient plus nombreux à rapporter avoir voulu *aller jusqu'au bout* de leurs réalisations, et un grand nombre d'entre eux ont rapporté avoir voulu *s'arrêter avant la fin* (Figure 68a). Quelques uns d'entre eux ont réussi à construire des structures 3D significatives (Figure 67).

Lors de la séance *Création de classes*, les participants étaient plus nombreux à rapporter avoir voulu *s'arrêter avant la fin*. Quelques uns ont affirmé avoir voulu *aller jusqu'au bout* (Figure 68b).

6.4.4 *Discussion*

À l'issue de chaque séance d'expérimentation *Utilisation d'objets* et *Création de classes*, les résultats obtenus montrent que les étudiants ont bien progressé après utilisation de PrOgO. En effet, pour chacune des deux séances, la moyenne des scores obtenus au



(a) Utilisation d'objets.

(b) Création de classes.

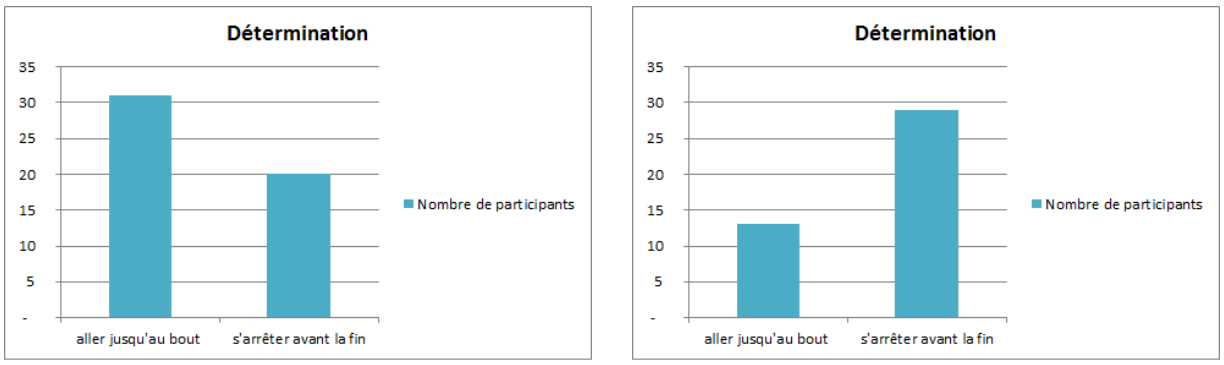
FIGURE 66 – Sentiment d'amusement chez les participants.

FIGURE 67 – Quelques réalisations 3D d'étudiants.

post-test est plus élevée que la moyenne des scores obtenus au pré-test. Les résultats ont également montré que les tests sont perçus moins difficiles par les étudiants après utilisation de PrOgO, ce qui confirme une amélioration dans leur compréhension des concepts abordés.

Les résultats d'analyse des traces ont montré un lien faible entre la différence des scores entre post-test et pré-test et les actions les plus représentatives de l'ensemble des participants. Étant donné ces résultats, il est difficile de conclure sur les actions et comportements des étudiants qui déterminent la différence des scores entre les post-tests et les pré-tests. Par ailleurs, les résultats d'analyse de la séance *Utilisation d'objets* ont montré que la différence des scores est plutôt liée (mais faiblement) à certaines actions réalisées dans l'éditeur de code. Ces actions comprennent la sélection de code, la modification de noms d'objets et la suppression de lignes de code. Les résultats d'analyse de la séance *Création de classes* ont montré que la différence des scores est le plus en lien (mais faiblement) avec certaines actions réalisées dans la scène 3D, telles que la modification de valeurs d'attributs. Cela indique qu'à la fois les actions réalisées dans l'éditeur de code et dans la scène 3D, influencent l'acquisition de connaissances.

Les points de vue des participants, concernant leurs sentiments d'amusement et de détermination, ont montré que les participants manquaient de motivation dans la réalisation de l'objectif du jeu. Ceci s'est traduit dans leurs interactions, dont les résultats d'analyse au travers de traces d'interaction étaient peu concluantes.



(a) Utilisation d'objets.

(b) Création de classes.

FIGURE 68 – Sentiment de détermination chez les participants.

6.4.5 *Conclusions*

Les résultats obtenus à l'issue de cette expérimentation ont montré que PrOgO est en mesure d'aider les étudiants débutants à acquérir de nouvelles connaissances. Cependant, il est difficile de conclure sur le type d'actions et les comportements adoptés par les étudiants qui sont à l'origine de cet apprentissage. De même les actions les plus représentatives des étudiants ne sont pas très concluantes.

Nous pensons, que le manque de signification des résultats d'analyse de traces d'interaction est dû, notamment, au contexte de l'expérimentation qui s'est déroulée à la fin de l'année universitaire. En effet, nous avions observé lors de l'expérimentation, que les étudiants n'étaient pas très motivés pour découvrir les fondamentaux de la POO. Cela a également été confirmé par leurs appréciations et opinions subjectives. Ce manque de motivation a influencé leurs attitudes, qui sont traduites dans leurs interactions avec l'interface de PrOgO.

Nous soulignons également les limites de notre méthodologie de collecte de données, qui est entièrement fondée sur des traces d'interaction constituées de l'historique des évènements des interactions des participants avec l'interface de PrOgO, dans un ordre chronologique de leurs actions. Avec cet historique d'évènements, il n'est pas possible de savoir à quel moment, quel participant regarde quel composant de l'interface. En effet, le fait de concentrer certaines actions dans l'éditeur de code ou dans la scène 3D, ne signifie pas nécessairement que l'utilisateur apprend exclusivement grâce au composant qu'il manipule. Étant donné que la scène 3D est synchronisée avec l'éditeur de code, l'apprentissage découle également des composants de l'interface que l'apprenant regarde à un moment donné.

6.5 CONCLUSIONS ET RÉSUMÉ DES CONTRIBUTIONS

La méthode de recueil des ressentis des participants et de leurs sentiments d'avoir appris via un questionnaire présente un manque de crédibilité. En effet, à l'issue de notre première expérimentation qui s'est basée sur un questionnaire sur les ressentis et points de vue, nous avons constaté qu'il était difficile d'estimer avec précision combien

les participants ont réellement amélioré leurs connaissances. En effet, un inconvénient majeur de cette méthode réside dans le fait qu'un participant puisse surestimer ou sous-estimer sa compréhension des notions évaluées.

La méthodologie basée sur le recueil et l'analyse de traces d'interaction numériques s'avère plus prometteuse. En effet, les résultats obtenus à l'issue de notre deuxième étude expérimentale, ont montré l'intérêt de cette méthodologie. Les techniques d'analyse statistiques visuelles, nous ont permis d'interpréter de façon formelle, l'usage que les participants ont fait de PrOgO. Les participants ont été catégorisés et caractérisés sur la base de leurs actions effectuées dans l'interface de PrOgO. Nous avons pu ressortir des indicateurs d'utilité, d'utilisabilité et d'acceptabilité.

Par ailleurs, nous avons conçu des tests de vérification de connaissances sur la base de la taxonomie de Bloom révisée et nous avons analysé les scores des participants en nous basant sur des indices statistiques. Les résultats d'analyse ont montré que PrOgO est en mesure de réussir en tant qu'environnement d'aide à l'apprentissage. Nous avons également tenté d'établir un lien entre une évaluation pré/post de connaissances et l'analyse de traces d'interaction. Les techniques d'analyse statistiques offrent un moyen de savoir formellement quelles sont les actions réalisées dans l'interface qui déterminent le changement de l'état des connaissances des participants. Bien que les techniques d'analyse le permettent, les résultats de notre dernière expérimentation n'ont pas été très concluants. Nous pensons que cela est dû au contexte d'expérimentation qui n'était pas très approprié. En effet, notre dernière étude expérimentale s'est déroulé à la fin de l'année universitaire.

Suite à nos études expérimentales, nous soulignons également les limites de notre méthodologie de collecte de données numériques, essentiellement fondée sur des traces d'interaction, donnant un historique d'évènements dans un ordre chronologique des actions de manipulation de l'interface. L'analyse complète des usages de PrOgO, nécessite de recueillir les informations relatives aux composants de l'interface sur lesquels est porté le regard de l'apprenant lors de son interaction avec PrOgO, en plus de ses actions réalisées dans l'interface. La nécessité de suivre le regard de l'apprenant découle notamment du fait que PrOgO dispose d'une scène 3D virtuelle entièrement synchronisée avec l'éditeur de code. Les résultats d'exécution des instructions de code sont instantanément visualisées dans l'interface 3D. Parallèlement, les actions réalisées dans la scène 3D sont interprétées instantanément en instructions de code. De ce fait, un apprenant qui enregistre un grand nombre d'actions dans la scène 3D est susceptible de regarder le code correspondant généré dans l'éditeur, sans qu'il enregistre des actions dans l'éditeur de code. De la même façon, un apprenant qui enregistre un grand nombre d'actions dans l'éditeur de code, a la possibilité de regarder les résultats d'exécution de ses instructions directement sur les graphiques 3D.

Certaines recherches actuelles s'orientent vers les techniques de suivi du regard (*eye-tracking*) (Conati et Merten, 2007), pour l'analyse des interactions utilisateurs dans le contexte d'environnements d'apprentissage. Nous pensons que leur utilisation dans l'évaluation des interactions avec PrOgO, peut apporter d'avantage d'informations relatives aux attitudes d'apprenants et améliorer notre méthodologie de recueil de données.

La première expérimentation est apparue dans la publication suivante :

Djelil, Fahima, Benjamin Albouy-Kissi, Adélaïde Albouy-Kissi, Eric Sanchez et Jean-Marc Lavest (2015). « Towards a 3D virtual game for learning Object-Oriented Programming fundamentals and C++ language - Theoretical considerations and empirical results ». In : *Proceedings of the 7th International Conference on Computer Supported Education*, p. 289–294.

CONCLUSION

Cette partie a pour but d'énoncer nos principales contributions dans ce travail de thèse, qui s'articulent autour de concepts clefs sur l'apprentissage par le jeu, et plus spécifiquement sur les micromondes de programmation. Nos contributions comprennent la conception d'un nouveau micromonde de POO qui met en œuvre les principes de conception et d'évaluation tirés de certains travaux existants. Cette partie donne également les questions de recherche sur lesquelles ce travail de thèse aboutit et s'achève avec notre positionnement théorique.

CONCLUSION

BILAN DES CONTRIBUTIONS

Nos contributions s'articulent, tout d'abord, autour des concepts clefs suivants :

— un premier concept est celui du *jeu-play* qui se distingue du concept du jeu-game. Une terminologie que nous reprenons des auteurs Sanchez, Emin-Martinez et Mandran (2015) et Shaffer (2006), qui considèrent que l'apprentissage est le résultat des interactions de l'apprenant avec le jeu-game. Ces interactions donnent forme à une situation, le jeu-play, dans laquelle les connaissances de l'apprenant se développent. Cette distinction nous a permis de situer l'apprentissage et d'identifier quelle partie du jeu, structure ludique ou interactions avec cette structure, il faut considérer lors de la conception et de l'évaluation des apprentissages. Notre point de vue d'un jeu-play n'est pas l'intégration d'éléments constitutifs de jeux vidéos dans un artefact informatique. Nous considérons que concevoir un jeu-play revient à concevoir une situation dans laquelle, l'apprenant est amené à exercer sa créativité et son imagination. Il faut veiller, néanmoins, à ce que certains attributs soient soigneusement intégrés dans la structure de l'artefact informatique, telles que les modalités de représentation qui doivent être conçues de manière fidèle au modèle de connaissances considéré. Les apprentissages émanant des interactions, leur évaluation peut être réalisée par l'analyse des interactions de l'apprenant avec l'artefact informatique.

— un second concept est une *approche didactique par emboîtement hiérarchique* des concepts fondamentaux de la POO, qui découle directement de l'approche Objets-D'abord (Object-First), qui a fait l'objet de nombreux travaux anglo-saxons relevant de la didactique de l'informatique. Cette approche ressort également des recherches menées par Bennedsen et Schulte, 2007, visant à comprendre son déroulement en pratique. Leur étude a abouti à l'identification de trois concepts structurant cette approche, à savoir : 1) utilisation d'objets, 2) création de classes, 3) création de modèles OO. Nous voyons dans cette approche un emboîtement de trois catégories de concepts. La première catégorie contenant une seconde, qui contient à son tour une troisième. Il s'agit à chaque étape d'amener l'apprenant à interagir avec une catégorie de concepts, les détails de chaque catégorie contenue étant occultée par une catégorie contenante. Les notions occultées sont introduites directement à l'étape suivante. Cela permet de s'affranchir de la complexité inhérente aux concepts OO, qui sont très fortement inter-liés.

— un troisième concept est celui de *système de représentation transitionnel*, formulation que nous proposons pour désigner le système de représentation propre à un micromonde de programmation. Cette formulation découle de l'expression "objets transitionnels" introduite par Papert (1980), puis reprise par Lawler (1987), pour désigner les éléments constitutifs d'un micromonde de programmation. Ce sont des objets visuels ou tangibles, interactifs et directement manipulables par l'ap-

prenant. Ils sont dits transitionnels car ils possèdent des propriétés communes, à la fois avec des concepts de programmation formels et des objets plus concrets de l'expérience sensible ou du monde réel. L'interaction de l'apprenant avec ces objets transitionnels, lui permet de développer des connaissances sur les concepts formels et abstraits que ces objets permettent de représenter. Nous étendons le concept d'objets transitionnels à celui de système de représentation transitionnel, car nous voyons dans un micromonde un ensemble organisé d'éléments conceptuels reliés entre eux, influençant la représentativité de ces objets transitionnels et définissant leur lien sémantique avec les concepts qu'ils représentent. Les fondements conceptuels que nous avons identifiés sont *la métaphore, la visualisation, le jeu-play et l'apprentissage*. Un micromonde de programmation emploie la métaphore pour décrire des concepts abstraits, pouvant être difficiles à comprendre, au moyen d'autres concepts familiers et intuitifs, donc faciles à comprendre. La métaphore est illustrée visuellement au travers des objets transitionnels dans un contexte concret et significatif imprégné de jeu-play. Le jeu-play découle notamment des actions créatives de l'apprenant sur l'interface du micromonde, qui répond par un feedback dynamique. Enfin, la métaphore, la visualisation et le jeu-play conduisent à la construction active des connaissances chez l'apprenant. Le jeu-play favorise les interactions avec les objets transitionnels, qui illustrent une métaphore visuelle par laquelle un lien sémantique est établi avec les concepts formels de la programmation.

Nos contributions comprennent également la conception d'un *framework* d'applications pour la gestion d'événements multi-touches tangibles et la découverte automatique du réseau.

— Le *framework Tactileo* est une contribution technique. Il offre une interface de développement d'applications Qt de haut niveau, pour la gestion d'événements utilisateurs multi-touches tangibles et la découverte automatique du réseau. Il fournit une librairie de développement qui constitue une aide pour le développement d'applications dédiées à une classe mutli-dispositifs tactile, telle que la classe Tactileo. Il permet l'interconnexion d'interfaces tactiles et d'autres dispositifs informatiques en synchronisant des ressources pédagogiques de manière transparente. Ce *framework* ne contribue pas à la recherche en Interaction Homme-Machine, il implémente certains protocoles fondés sur d'autres *frameworks* existants, et son architecture conceptuelle est semblable à celle d'autres *frameworks* existants dont il s'inspire.

Une contribution majeure de cette thèse est la conception de l'environnement PrOgO, que nous définissons comme un nouveau micromonde de POO fondé sur un jeu de construction et d'animation 3D.

— PrOgO met en œuvre l'ensemble des concepts issus de notre étude bibliographique portée sur l'apprentissage de la programmation par le jeu. Premièrement, PrOgO est conçu de façon à mettre en place un jeu-play qui découle des interac-

tions de l'apprenant avec son interface. Jouer avec PrOgO consiste à imaginer et à créer des constructions 3D significatives et les animer. Deuxièmement, PrOgO implémente un système de représentation transitionnel, au sein duquel des concepts fondamentaux de la POO sont représentés par des composants graphiques 3D visuels et interactifs. L'interface de PrOgO comprend un environnement virtuel restreint qui implémente un système de rendu 3D et des techniques d'interaction répandues dans les applications de RV. Les graphiques 3D qui se créent et se manipulent à l'interface sont des objets transitionnels, qui illustrent une métaphore de construction et d'animation, décrivant de manière fidèle et consistante les concepts de POO. PrOgO synchronise l'environnement 3D interactif avec un éditeur de code à autocomplétion. Les composants graphiques dans la scène 3D ont des propriétés communes à celles d'objets informatiques, et les constructions 3D réalisées sont des représentations visuelles de systèmes OO. Les actions directes de l'apprenant sur les graphiques 3D sont traduites en instructions de code C++, et les résultats des actions d'autocomplétion sont instantanément visibles dans la scène 3D. L'apprenant construit ses connaissances en produisant du sens sur les concepts OO, suite à ses interactions avec l'interface de PrOgO. Troisièmement, PrOgO implémente une *approche didactique par emboîtement hiérarchique* des concepts fondamentaux de la POO à travers deux modes. Un premier mode se consacre à l'utilisation d'objets, dans lequel l'apprenant est amené à développer des connaissances en lien avec la notion d'objet. Un second mode se consacre à la création de classe, dans lequel l'apprenant accède à des notions en lien avec la classe, un concept volontairement occulté dans le premier mode. Cette approche vise un apprentissage progressif de ces notions, qui sont étroitement liées les unes aux autres. PrOgO est également conçu de façon à être utilisé dans la classe Tactileo. Il implémente les interfaces de gestion d'évènement tactiles et de découverte automatique du réseau, fournies par le *framework Tactileo*. Enfin, PrOgO permet des études expérimentales de ses usages, en générant des traces d'interaction numériques. La conception de PrOgO contribue aux avancées théoriques sur les dimensions de conception des micromondes de programmation, et en particulier des micromondes de POO.

Une autre contribution importante de cette thèse est l'utilisation de techniques d'analyse de l'apprentissage dans l'évaluation des usages de PrOgO. Le but étant d'examiner son potentiel sur l'apprentissage, que nous situons dans les interactions qui se nouent ente l'apprenant et l'interface de PrOgO lors du jeu-play.

— À l'issue de notre étude bibliographique, nous avons adopté l'idée qu'un apprenant construit ses connaissances pendant le jeu-play, lorsqu'il interagit avec l'artefact informatique qui rend possible ce jeu-play. Par conséquent, l'évaluation des apprentissages avec PrOgO se traduit par l'analyse des interactions de l'apprenant avec son interface. Notre méthodologie d'évaluation relève du domaine de l'analyse de l'apprentissage par le recueil et l'analyse de traces d'interaction numériques. PrOgO est conçu de façon à permettre le recueil de traces d'interaction, par la génération de l'historique des actions de l'apprenant dans un ordre chronologique. Les méthodes d'analyse statistique ACP et CAH nous ont permis

d'évaluer l'usage de PrOgO, par la classification et la caractérisation d'apprenants. Cela nous a permis de déduire que PrOgO permet la mise en place d'un jeu-play à partir duquel émerge l'apprentissage. Les classes ressorties ont montré, que les apprenants acceptent de jouer, et qu'ils adoptent des comportements différents lors de leur interaction avec PrOgO et donc avec les concepts considérés. Les étudiants produisent du sens sur les concepts de la POO, en adoptants des comportements différents. En complément à l'analyse statistique des actions, nous examinons l'état des connaissances d'apprenants, au travers de tests de connaissances élaborés sur la base des trois premiers niveaux de la Taxonomie de Bloom révisée des objectifs d'apprentissage. Les évaluations pré/post ont montré que PrOgO peut aider les débutants à acquérir de nouvelles connaissances. Bien que les résultats de notre dernière expérimentation n'aient pas été très concluants, étant donné les circonstances de cette expérimentation, nous avons vu que l'analyse statistique des actions au travers de traces d'interaction, offre un moyen de connaître de manière fine, les actions formulées qui déterminent la progression des scores obtenus aux tests.

Notre contribution par cette méthodologie d'évaluation, se présente comme une avancée dans les méthodologies d'évaluation des micromondes de programmation, qui sont restées longtemps fondées sur l'utilisation de questionnaires pour le recueil des points de vue des participants, l'observation de leurs comportements en présentiel et les évaluations pré/post de l'acquisition des connaissances. Les méthodes basées sur les questionnaires et les observations directes de comportements manquent de crédibilité, et les évaluations pré/post de l'acquisition des connaissances ne donnent pas d'indications sur la nature des interactions et leur rôle dans la progression des connaissances. Les études expérimentales menées avec PrOgO, relevant de l'analyse de l'apprentissage, marquent un début dans l'avancée des démarches méthodologiques d'évaluation de micromondes, notamment dans la démonstration de leur potentiel sur l'apprentissage de la programmation.

PERSPECTIVES DE RECHERCHE ET QUESTIONS OUVERTES

Nos perspectives de recherche s'articulent autour du concept de micromondes de programmation. Il s'agit de travailler à des avancées théoriques et méthodologiques sur les questions de conception et d'évaluation de micromondes pour l'apprentissage de la programmation et les implications de ces avancées sur la discipline de l'Informatique.

La question de conception implique de considérer le micromonde, comme un système de représentation transitionnel au sein duquel les concepts formels de programmation sont décrits au moyen de métaphores. Se pose alors la question de la fidélité et de la complétude des métaphores employées dans la représentation des concepts formels. La conception d'un micromonde nécessite également de considérer le jeu-play qui découle des interactions de l'apprenant avec le micromonde. Lors du jeu-play, l'apprenant construit ses apprentissages, en produisant du sens sur ce qu'il manipule et observe dans l'interface du micromonde. De ce point de vue, des avancées méthodologiques sont nécessaires pour examiner le rôle de ces interactions sur l'apprentissage.

Les méthodologies d'analyse de l'apprentissage (*learning analytics*) incluant la collecte, l'analyse et la production de rapports de données sur les apprenants et leurs contextes d'apprentissage offrent de réelles opportunités.

L'utilisation des micromondes de programmation nécessite d'examiner l'expérience d'apprentissage dans sa globalité. De ce fait, il est nécessaire d'explorer comment les micromondes et les techniques d'analyse statistique peuvent améliorer et promouvoir l'apprentissage de la programmation, et de l'Informatique en général. En particulier, il s'agit de considérer les questions suivantes [3] :

1. Vers quelles modalités d'interaction et interfaces utilisateurs tendent les micromondes de programmation ?
2. Quelles sont les données qui peuvent être collectées des situations d'apprentissage créées par les micromondes de programmation ?
3. Comment les techniques d'analyse de donnés et les techniques de visualisation, peuvent nourrir la réflexion et offrir des perspectives à la fois pour l'apprenant, l'enseignant ou le chercheur ?
4. Quelles sont les avancées théoriques et pratiques que les techniques de suivi du regard combinées avec l'analyse de l'apprentissage peuvent apporter aux micromondes ?
5. Comment l'usage de micromondes combiné avec l'analyse de l'apprentissage peut aider à promouvoir l'enseignement de la programmation ?
6. Quels sont les implications de ces avancées sur d'autres domaines de l'Informatique ?
7. Quelles sont les implications sur l'avenir des langages de programmation et l'Informatique en général ?

POSITIONNEMENT THÉORIQUE

Ce travail de thèse se situe à l'intersection de plusieurs disciplines. Tout d'abord il se situe au cœur du domaine de l'Informatique, car centré sur la problématique de l'apprentissage de la POO, l'une des difficultés conceptuelles rencontrées consiste à établir un lien sémantique entre des concepts de POO formels, et des représentations visuelles informelles. Cette correspondance est établie au sein d'un système transitionnel au travers d'une métaphore de construction et d'animation 3D. Cette métaphore décrit de manière fidèle les concepts d'objets, de classe, de relations d'interaction entre classes et de systèmes OO. Du point de vue de l'ingénierie des langages, ce travail s'apparente à la définition d'un Langage de Spécification de Domaine (*Domain Specification Language* -

3. Une première tentative visant à aborder ces questions et à soulever d'autres questions pouvant contribuer à améliorer les pratiques et les recherches actuelles, est notre implication dans l'organisation d'un premier atelier francophone sur l'apprentissage instrumenté de l'Informatique : https://apprentissageinstrumentdelinformatique.wordpress.com. Cet atelier tend à réunir les travaux actuels portés sur les outils et environnements émergents visant à favoriser l'apprentissage de l'Informatique.

DSL) pour PrOgO. En effet l'environnement PrOgO définit une syntaxe concrète constituée de l'ensemble d'entités visuelles présentes à l'interface (objets transitionnels). Cette syntaxe est sémantiquement liée à une syntaxe abstraite, qui n'est pas intégrée au sein de l'environnement PrOgO, bien qu'elle soit définie de façon à décrire les concepts illustrés dans son interface (notation UML). PrOgO traduit sa syntaxe concrète en code C++ et dispose d'un analyseur syntaxique fondé sur l'analyse des expressions régulières. Cet analyseur interprète un programme entré par autocomplétion à l'interface, et le résultat de son exécution se traduit dans sa syntaxe concrète.

Ce travail de thèse emprunte également au domaine de la RV, car il concerne un micromonde intégrant un environnent virtuel restreint et interactif, qui implémente des techniques d'interaction 3D répandues dans les applications de RV. Il empreinte également à la robotique pour l'éducation, car l'objet du jeu-play consiste à construire et à animer des structures robotiques. Le but étant de développer des connaissances OO représentées dans et par ces structures.

Ce travail est en lien avec la didactique de l'Informatique, car il traite des questions liées à l'enseignement de la programmation, notamment par l'implémentation d'une nouvelle approche didactique de l'OO, tirée des recherches effectuées dans ce domaine dans la communauté anglo-saxonne. Il emprunte également aux sciences de l'éducation, car il considère l'apprentissage qui se construit lors de situations de jeu-play. L'évaluation de l'apprentissage se traduit par l'analyse des usages et des interactions qui se nouent entre l'apprenant et l'artefact informatique lors du jeu-play. Enfin, ce travail emprunte aux STICs et s'inscrit dans la discipline de l'ingénierie des EIAHs, car il traite de la conception d'un outil numérique à visée éducative, ayant des implications sur l'apprentissage et l'enseignement, répondant à un besoin de renouvellement de pratiques et d'approches pédagogiques.

ANNEXES

A

PLUGINS DE DÉVELOPPEMENT DU FRAMEWORK TACTILEO

A.1 INTERFACES DE SOUSCRIPTION ET DE TRAITEMENT D'ÉVÈNEMENTS MULTI-TOUCHES ET TANGIBLES

A.1.1 *Plugin de gestion d'évènements multi-touches et tangibles*

Le plugin de gestion d'évènements multi-touches et tangibles est fourni comme une interface permettant la souscription aux évènements multi-touches et tangibles. Elle dépend d'autre interfaces qui permettent la récupération de propriétés des différents évènements (Figure 69).

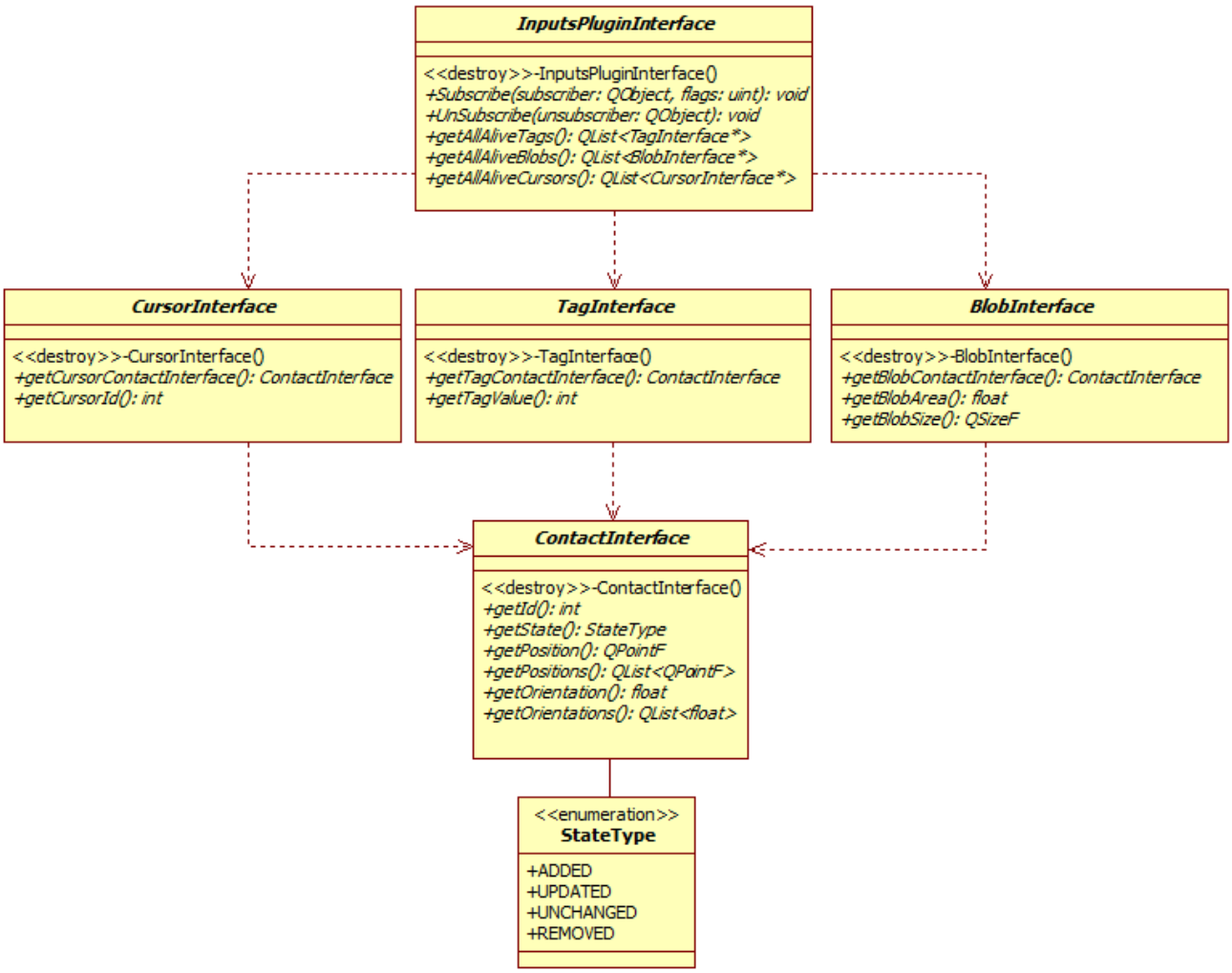


FIGURE 69 – Diagramme de classes UML décrivant les interfaces du plugin de gestion d'évènements multi-touches et tangibles.

A.1.2 *Template de spécification de type d'évènements à gérer*

Le framework Tactileo fournit le template *InputEvent*. Il permet à une application souscrivant au framework de spécifier quels types d'évènements multi-touches et tangibles elle souhaite gérer (Figure 70).

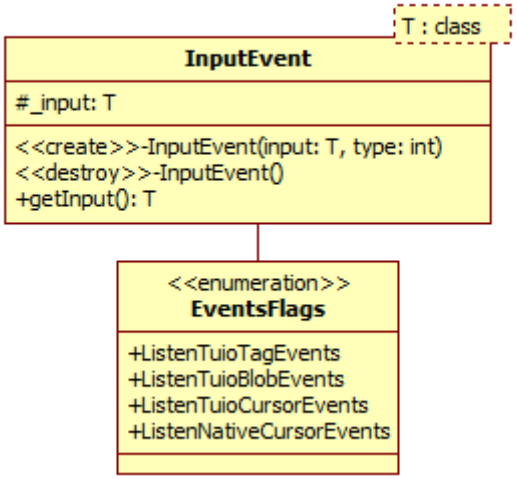


FIGURE 70 – Le template InputEvent.

A.2 PLUGIN DE SOUSCRIPTION ET DE GESTION AUTOMATIQUE DU RÉSEAU

Le plugin de découverte automatique du réseau est une interface permettant la souscription au réseau. Elle dépend d'une autre interface qui permet de fournir des informations relatives aux dispositifs souscripteurs (Figure 71).

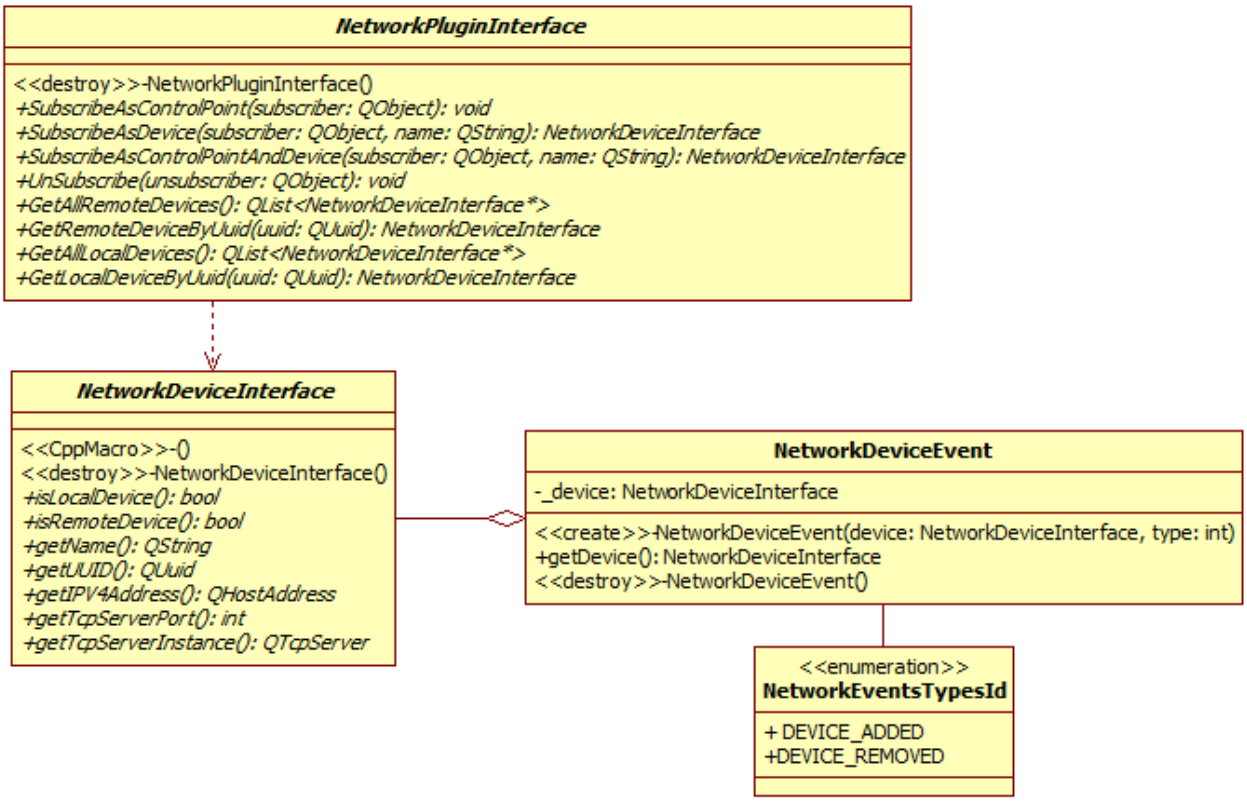


FIGURE 71 – Diagramme de classes UML décrivant les interfaces du plugin de découverte automatique du réseau.

B

DESCRIPTION DES PROPRIÉTÉS GÉOMÉTRIQUES D'UN COMPOSANT 3D DANS PROGO

Un objet 3D dans PrOgO est modélisé avec *3DsMax* puis exporté dans un fichier XML à l'aide de *OgreMax* par souci de compatibilité avec *Ogre*. Le fichier XML contient les informations géométriques de l'objet modélisé dans *3DsMax*, à savoir son maillage (*mesh*), son matériau, sa position, des valeurs de mise à l'échelle et sa rotation dans la scène de modélisation de *3DsMax*. Un objets 3D dans PrOgO est défini par sa géométrie, sa position (x,y,z), sa taille et son orientation, et n'a pas de matériau. A ces informations sont ajoutées des informations de propriétés de la classe que l'objet 3D représente, à savoir un attribut couleur et un attribut angle de rotation.

Un fichier XML décrivant un objet 3D dans PrOgO possède la structure suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<scene formatVersion="1.0" upAxis="z" unitsPerMeter="39.3701" minOgreVersion="1.9"
    ogreMaxVersion="2.6.4" author="OgreMax Scene Exporter (www.ogremax.com)"
    application="3DS Max">
    <classProperties canRotate="bool" canChangeColor="bool"/>
        <nodes>
        <node name="node1">
            <position x="valueX" y="valueY" z="valueZ"/>
            <scale x="valueX" y="valueY" z="valueZ"/>
            <rotation qx="valueQX" qy="valueQY" qz="valueQZ" qw="valueQW"/>
            <entity name="geometryName" castShadows="true" receiveShadows="true"
                meshFile="meshName.mesh">
                <subentities>
                    <subentity index="0" materialName="NoMaterial"/>
                </subentities>
            </entity>
        </node>
        <node name="node2">
        ...
        </node>
        ...
    </nodes>
</scene>
```

A titre d'exemple, l'objet 3D *Cube* est décrit par le fichier XML suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<scene formatVersion="1.0" upAxis="z" unitsPerMeter="39.3701" minOgreVersion="1.9"
    ogreMaxVersion="2.6.4" author="OgreMax Scene Exporter (www.ogremax.com)"
    application="3DS Max">
    <classProperties canRotate="false" canChangeColor="true"/>
        <nodes>
```

```
        <node name="connector_000">
            <position x="0" y="0" z="1.29097e-009"/>
            <scale x="1" y="1" z="1"/>
            <rotation qx="-0.183013" qy="-0.683013" qz="-0.183013" qw="0.683013"/>
            <entity name="connector_000" castShadows="true" receiveShadows="true"
                meshFile="connector.mesh">
                <subentities>
                    <subentity index="0" materialName="connector"/>
                </subentities>
            </entity>
        </node>
        <node name="connector_001">
            <position x="0.564242" y="-0.983432" z="0"/>
            <scale x="1" y="1" z="1"/>
            <rotation qx="0.5" qy="-0.5" qz="0.5" qw="0.5"/>
            <entity name="connector_001" castShadows="true" receiveShadows="true"
                meshFile="connector.mesh">
                <subentities>
                    <subentity index="0" materialName="connector"/>
                </subentities>
            </entity>
        </node>
        <node name="connector_002">
            <position x="1.11674" y="-0.0184772" z="-7.85837e-010"/>
            <scale x="1" y="1" z="1"/>
            <rotation qx="0.683013" qy="0.183013" qz="0.683013" qw="-0.183013"/>
            <entity name="connector_002" castShadows="true" receiveShadows="true"
                meshFile="connector.mesh">
                <subentities>
                    <subentity index="0" materialName="connector"/>
                </subentities>
            </entity>
        </node>
        <node name="geometry_000">
            <position x="0.564242" y="-0.386263" z="0"/>
            <scale x="1" y="1" z="1"/>
            <rotation qx="5.13515e-009" qy="1.53039e-016" qz="2.98023e-008" qw
                ="1"/>
            <entity name="geometry_000" castShadows="true" receiveShadows="true"
                meshFile="BifurcationY_geometry_000.mesh">
                <subentities>
                    <subentity index="0" materialName="NoMaterial"/>
                </subentities>
            </entity>
        </node>
    </nodes>
</scene>
```

C

INTERFACES UTILISATEURS DES DIFFÉRENTS PROTOTYPES DE PROGO

C.1 INTERFACE DU PROTOTYPE EXPÉRIMENTAL DE PROGO

L'interface du prototype expérimental de PrOgO dispose en son centre d'une scène 3D invitant l'utilisateur à y placer des composants 3D représentant des objets informatiques (Figure 72). En bas de l'interface se trouvent les composants 3D représentant les classes instanciables. A droite de l'interface se trouve l'onglet de code, dans lequel sont générées les instructions C++ correspondantes aux actions directes de l'utilisateur sur les composants 3D.

L'onglet de code `main.cpp` contient la déclaration du programme principal `main`. Le programme principale est vide et correspond à l'état initial de la scène 3D.

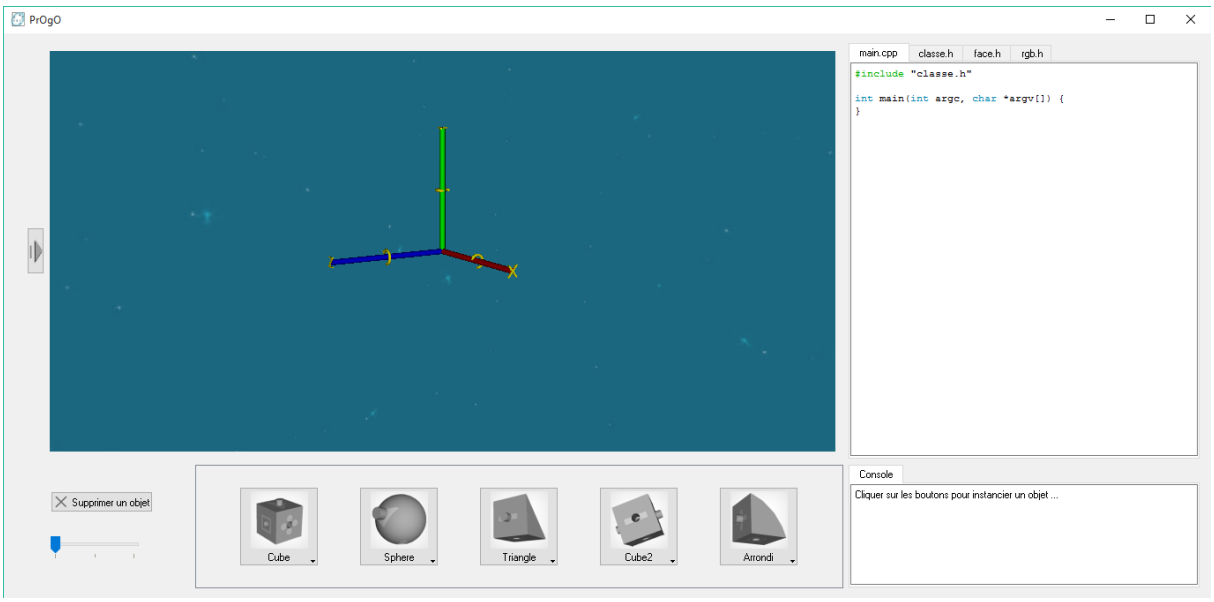


FIGURE 72 – Interface principale du prototype expérimental de PrOgO.

C.2 INTERFACE UTILISATEUR DE PROGO V.0.6

L'interface utilisateur de PrOgO v.0.6 dispose principalement d'une scène 3D interactive et d'un éditeur à autocomplétion. La version 0.6 de PrOgO permet l'utilisation d'objets et l'édition de code C++, mais ne permet pas la création de classes (Figure 73).

Cinq classes sont proposées à l'interface comme modèle d'objets, à savoir `Base, Cube, BifurcationY, Engrenage et Roue`. L'objectif étant de construire et d'animer des structures robotiques ou mécaniques en 3D. Les actions directes de l'utilisateur sur les composants graphiques sont traduites en instructions de code qui sont générées dans l'édi-

teur à autocomplétion. Les instructions de code entrées en autocomplétion sont directement visibles dans la scène 3D, sur les composants graphiques correspondants.

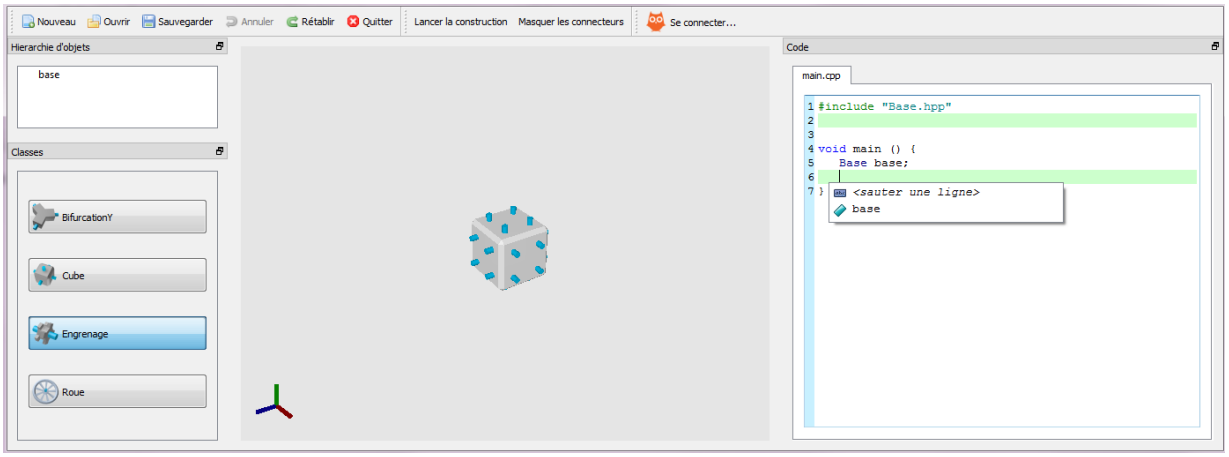


FIGURE 73 – Interface principale de PrOgO v.0.6.

C.3 INTERFACE UTILISATEUR DE PROGO V.0.7

L'interface de PrOgO v.0.7 offre toutes les fonctionnalités de la version 0.6 et étend cette dernière vers la création de nouvelles classes. Toute construction 3D dans PrOgO est une nouvelle classe que l'apprenant peut ajouter à sa liste de classes. Quand cette classe est créée, l'apprenant a accès à son fichier de déclaration et à son fichier de définition. L'apprenant peut également l'utiliser en l'instanciant directement dans la scène 3D, ou en entrant des instructions de code en autocomplétion dans l'éditeur de code (Figure 74).

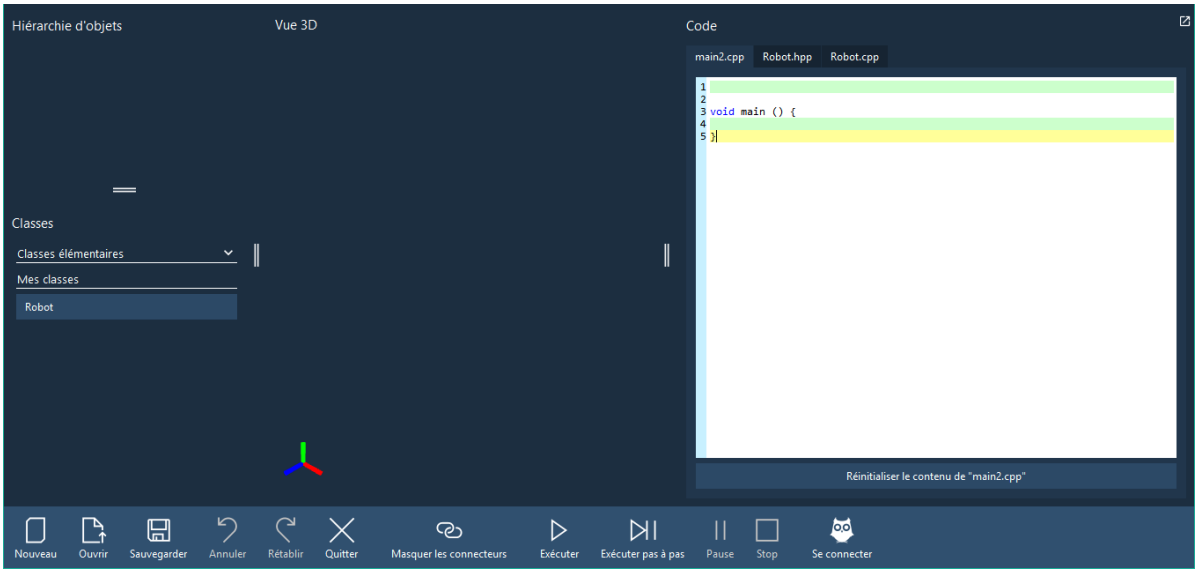


FIGURE 74 – Interface de PrOgO v.0.7 montrant le mode *Création de classe*.

D

TRACES D'INTERACTION NUMÉRIQUES GÉNÉRÉES PAR PROGO

Les traces d'interaction générées par PrOgO sauvegardent pour chaque utilisateur l'historique des évènements de son interaction avec l'interface de PrOgO, dans un ordre chronologique des actions. Un fichier est généré pour chaque participant, comportant l'identifiant de l'utilisateur, le *timestamp* de l'action (le moment de réalisation de l'action), l'identifiant de l'action, l'intitulé de l'action et les paramètres de l'action (Tableau 27).

Champ	Signification
timestamp	moment de réalisation de l'action
idUtilisateur	identifiant de l'utilisateur réalisant l'action
idAction	identifiant de l'action
action	intitulé de l'action
paramètres	paramètres de l'action

Tableau 27 – Modèle de traces propre à PrOgO.

Les actions tracées par les deux versions de PrOgO sont données dans le Tableau 28.

idAction	action	paramètres	PrOgO v.0.6	PrOgO v.0.7
0	start	none	+	+
1	instanciate	objectName$objectClassName	+	+
2	setAttributeColor	objectName$colorCode	+	+
3	setAttributeRotationAngle	objectName$angle	+	+
4	callFuncConnect	object1Name$object1dConnector$object2Name $object2dConnector	+	+
5	callFuncTurnFor	objectName$angle$seconds	+	+
6	callFuncColorFor	objectName$colorCode$seconds	+	+
7	completeCode	posInCode$completedCode	+	+
8	selectCode	selectedText | none	+	+
9	undo	none	+	+
10	redo	none	+	+
11	openProject	fileName.progo$successOpeningFile	+	+
12	newProject	none	+	+
13	saveProject	fileName.progo	+	+
14	end	none	+	+
15	hierarchySelectedItem	SelectedItem	-	+
16	popHelpDialog	BoxTitle	-	+
17	closeHelpDialog	BoxTitle$true (if box toogled) | false (otherwise)	-	+
18	startSimulation	true(if step by step) | false (otherwise)	-	+
19	stepSimulation	none	-	+
20	autoPauseSimulation	none	-	+
21	pauseSimulation	none	-	+
22	resumeSimulation	none	-	+
23	endSimulation	none	-	+
24	stopSimulation	none	-	+
25	refactorVariable	oldVariable$newVariable$objectClassName	-	+
26	deleteLine	posInCode$lineNumber$codeFileName	-	+
27	createUserClass	ClassName	-	+
28	instanciateUserClass	ClassName$instanceName	-	+
29	switchUIToLvl1	none	-	+
30	switchUIToLvl2	none	-	+
31	tactileo/connect	username$tactileoID	-	+
32	tactileo/save	PrivacyMode$path	-	+
33	tactileo/load	PrivacyMode$path	-	+
34	network/monitoringEnabled	true | false	-	+
35	network/allowedSession	none	-	+
36	network/allowedLvl2	none	-	+

les symboles + et - indiquent que l'action est tracée ou non tracée dans la version de PrOgO correspondante.

Tableau 28 – Actions tracées dans les versions 0.6 et 0.7 de PrOgO.

E

TAXONOMIE DE BLOOM RÉVISÉE

Dans l'élaboration de tests de vérification des connaissances, nous nous sommes appuyés sur la taxonomie de Bloom révisée des objectifs éducatifs. Cette taxonomie a été d'une grande utilité pour une grande majorité d'éducateurs, dans la définition d'activités d'apprentissage en contexte de classe (Anderson et Krathwohl, 2001). L'idée d'utiliser la taxonomie de Bloom révisée pour l'évaluation d'un jeu éducatif n'est pas nouvelle. Le jeu *X-MED*, dont l'objectif pédagogique est de renforcer les concepts de mesure des systèmes logiciels, a été en partie évalué en utilisant des tests qui ont été élaborés en s'appuyant sur cette taxonomie (Von Wangenheim, Thiry et Kochanski, 2009).

La taxonomie de Bloom révisée organise de façon hiérarchique les objectifs d'apprentissage cognitifs en six niveaux distincts (Figure 75). Chaque niveau est plus sophistiqué qu'un niveau directement inférieur et requiert plus de compétences cognitives (Anderson et Krathwohl, 2001). Le premier niveau inférieur de la hiérarchie, *Reconnaître*, se réfère à la capacité d'identifier et de nommer des faits, des concepts ou des principes. Le second niveau directement supérieur, *Comprendre*, sous-entend qu'un apprenant est capable de construire du sens à partir de ce qu'il connaît et d'utiliser ses propres mots pour formuler cette connaissance. Le troisième niveau, *Appliquer*, suppose que l'apprenant connaît et comprend quelque chose avant de l'utiliser. Au quatrième niveau, *Analyser*, l'apprenant est amené à décomposer un problème dans ses composants et de discuter séparément chaque composant. Au cinquième niveau, *Évaluer*, l'apprenant peut être amené à effectuer des jugements concernant quelque chose en utilisant certains critères. Au sixième et dernier niveau, *Créer*, l'apprenant est amené à utiliser la connaissance apprise afin de former un tout cohérent ou de créer de nouveaux produits.

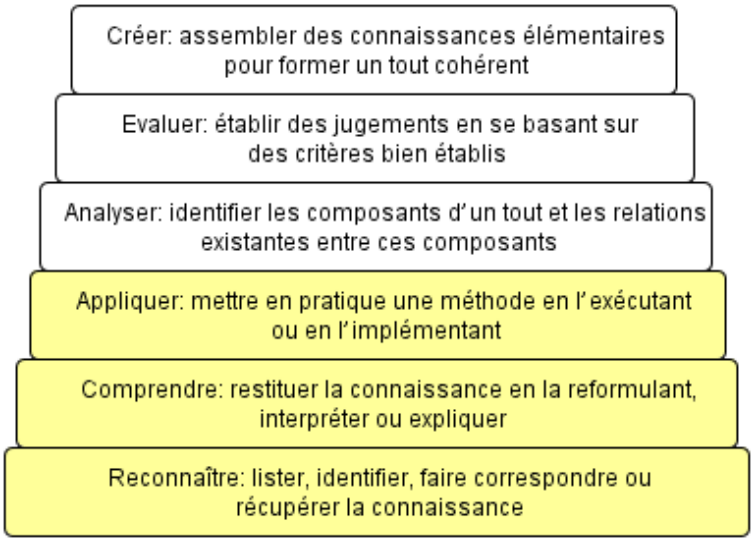


Figure 75 – Taxonomie de Bloom révisée des objectifs d'apprentissage.

F

TESTS DE VÉRIFICATION DES CONNAISSANCES ET QUESTIONNAIRES

F.1 TEST DE VÉRIFICATION DE CONNAISSANCES PORTÉ SUR L'UTILISATION D'OBJETS

Ce test a été utilisé pour l'évaluation de l'état des connaissances d'apprenants lors de l'expérimentation réalisée avec PrOgO v.0.6. Il a été ensuite réutilisé comme pré/post test lors de l'expérimentation réalisée avec PrOgO v.0.7. Son objectif est de vérifier des connaissances en lien avec la notion d'objet.

Le test est accompagné des codes `A` et `B` listés plus loin. Il contient les questions suivantes :

1. Dans le code A, listez les objets par leurs noms :
   .....
   .....
2. Identifiez la classe de chaque objet listé précédemment :
   .....
   .....
3. A quoi correspond chacune des instructions des lignes (13,18,32) :
   Se rapporter au code A [une seule réponse possible].
   (A) à un appel de méthode
   (B) à un changement de valeur d'attribut
   (C) à une instanciation de classe
4. A quoi correspond chacune des instructions des lignes (22,26,34) :
   Se rapporter au code A [une seule réponse possible].
   (A) à un appel de méthode
   (B) à un changement de valeur d'attribut
   (C) à une instanciation de classe
5. A quoi correspond chacune des instructions des lignes (12,25,31) :
   Se rapporter au code A [une seule réponse possible].
   (A) à un appel de méthode des objets :`engrenage_0, maBifurcation, maRoue`
   (B) à un appel de méthode des objets : `base, bifurcation_0, cube_0`
   (C) à un appel de méthode entre les objets `engrenage_0, maBifurcation, maRoue` d'une part et les objets `base, bifurcation_0, cube_0` d'autre part

6. Pour chaque objet identifié, listez les numéros de lignes des instructions qui agissent sur la modification de son état.
   Se rapporter au code A.
7. Cochez la ou les bonnes réponse(s).
   Se rapporter au code A. Cela concerne le bloc d'instructions `main()` [Plusieurs réponses possibles].

   (A) deux objets d'une même classe peuvent avoir le même nom

   (B) deux objets de différentes classes peuvent avoir le même nom

   (C) un objet possède un nom qui l'identifie de façon unique et qu'il ne partage pas avec d'autres objets

8. Un objet peut-il exister sans classe ?
   [Une seule réponse possible].

   (A) Oui

   (B) Non

9. Cochez la ou les bonnes réponse(s) :
   [Plusieurs réponses possibles.]

   (A) un attribut sert à l'objet d'un élément constitutif qui le distingue des autres objets

   (B) une méthode désigne pour l'objet l'un de ses savoirs faire

   (C) les attributs et les méthodes d'un objet servent à constituer son état

10. Comment peut-on modifier l'état d'un objet ?
    .....
    .....
11. Que signifient les écritures des lignes (1 et 2)
    Se rapporter au code B.
    .....
    .....
12. Que signifient les écritures des lignes (4 et 5)
    Se rapporter au code B.
    .....
    .....
13. Que signifient les écritures des lignes (7, 8, 9, 10 et 11)
    Se rapporter au code B.
    .....
    .....

14. Soit la classe `Charniere`, contenant les mêmes attributs et méthodes de la classe `Engrenage` dans le jeu PrOgO, créez une instance de cette classe et nommez la `maCharniere`.
    Implémentez l'instruction en respectant la syntaxe C++
    .....
15. Changez en Bleu (#0000ff) la couleur de l'objet `maCharniere`
    Implémentez l'instruction en respectant la syntaxe C++
    .....
16. Coloriez en Rouge (#ff0000) pendant 2 secondes l'objet `maCharniere`
    Implémentez l'instruction en respectant la syntaxe C++
    .....
17. Tournez l'objet `maCharniere` d'un angle de `90°`
    Implémentez l'instruction en respectant la syntaxe C++
    .....
18. Tournez l'objet maCharniere d'un angle de `90°` pendant 5 secondes
    Implémentez l'instruction en respectant la syntaxe C++
    .....

CODE A. Le code suivant est généré par PrOgO. Il est fourni afin de répondre aux questions (1, 2, 3, 4, 5, 6, 7) du test :

```
#include "Base.hpp"
#include "Engrenage.hpp"
#include "Cube.hpp"
#include "BifurcationY.hpp"
#include "Roue.hpp"

void main () {

    Base base;

    Engrenage engrenage_0;
    engrenage_0.connecter(0,base,5);
    engrenage_0.couleur = "#c09bff";

    Cube cube_0;
    cube_0.connecter(0,engrenage_0, 1);
    cube_0.couleur = "#b3ff4f";
    engrenage_0.angleDeRotation = 135;

    BifurcationY bifurcationy_0;
    bifurcationy_0.connecter(0,cube_0,1);
    bifurcationy_0.colorierPendant("#ffa9b2",5);

    BifurcationY maBifurcation;
```

```
    maBifurcation.connecter(0, bifurcationy_0,1);
    maBifurcation.colorierPendant("#ffa9b2",5);

    engrenage_0.tournerPendant(180,5);

    Roue maRoue;
    maRoue.connecter( 0,cube_0,5);
    maRoue.couleur = "#7e80ff" ;
    maRoue.angleDeRotation = 180;
    maRoue.tournerPendant(150,3);
}
```

CODE B. Ces écritures sont fournies afin de répondre aux questions (11,12,13) du test :

```
Cls obj1;
Cls obj2;

Obj1.att= val;
Obj2.att= val;

Obj2.meth(p);
obj1.meth1(p1,obj2,p2);
obj1.meth3(p1,p2);
obj1.meth4(p1,obj2);
obj2.meth5(obj2);
```

F.2 TEST DE VÉRIFICATION DE CONNAISSANCES PORTÉ SUR LA CRÉATION DE CLASSES

Ce test a été utilisé pour l'évaluation pré/post des connaissances d'apprenants lors de l'expérimentation réalisée avec PrOgO v.0.7. Son objectif est de vérifier des connaissances en lien avec la notion de classe.

Le test est accompagné du code `A`, `B` et `C` listés plus loin. Il contient les questions suivantes :

1. Il existe dans les fractions de codes (A,B,C) un nouveau type de données défini par le programmeur.

   Sélectionnez une réponse :

   (A) Vrai

   (B) Faux

   Si Vrai, donnez le nom de ce nouveau type de données :

   .....

2. Lesquels des codes A, B et C représentent une déclaration de classe, une définition de classe et une utilisation de classe :

   (A) Déclaration de classe

— Code A
— Code B
— Code C

(B) Définition de classe
— Code A
— Code B
— Code C

(C) Utilisation de classe
— Code A
— Code B
— Code C

3. Listez les mots clés utilisés pour implémenter l'encapsulation dans une classe. Se rapporter au code A, B et C.

.....

4. Que représente la méthode décrite aux lignes 19 du code (A) et 3 du code (B)

.....

.....

5. A quoi sert de créer de nouvelles classes ?

.....

.....

6. Les membres d'une classe regroupent :
Veuillez choisir une réponse :

(A) ses attributs et ses méthodes

(B) ses attributs uniquement

(C) ses méthodes uniquement

7. Quelle est la différence entre une déclaration d'une classe et une définition de classe ?

.....

.....

8. Veuillez choisir au moins une réponse :

(A) Les membres précédés du mot clé public sont accessibles à la fois par les autres membres de la même classe et en dehors de cette classe

(B) Les membres précédés du mot clé public sont accessibles uniquement en dehors de leur classe

(C) Les membres précédés par le mot clé private sont accessibles uniquement par les autres membres de la même classe.

(D) Les membres précédés par le mot clé private sont accessibles à la fois par les autres membres de la même classe et en dehors de cette classe.

9. Quel est le rôle de l'encapsulation dans une classe ?

   .....

   .....

10. Quel est le rôle du constructeur dans une classe ?

    .....

    .....

11. Un constructeur d'une classe se caractérise comme suit :
    Veuillez choisir au moins une réponse :

    (A) Doit porter le même nom que sa classe

    (B) Doit posséder un type de retour

    (C) Ni l'un ni l'autre

12. Décrivez les trois étapes d'utilisation d'une nouvelle classe

    .....

    .....

13. Vous disposez de deux classes préalablement implémentées `Cube et Roue`.
    La classe Roue est munie des deux méthodes :
    — `void connecter (int id_ConectRoue, Cube & cube, int id_ConectCube)` : sert à lier une roue sur son connecteur `id_ConectRoue` à un cube, sur son connecteur `id_ConectCube`.
    — `void tournerPendant(int angle, int Nb_Sec)` : sert à tourner une roue d'un angle (`angle`) pendant une durée de (`Nb_Sec`) secondes.

    Réalisez une classe `Charrette` constituée d'un cube et de deux roues. On prévoit :
    — Trois données membres privées : `cube, roue_0 et roue_1`, où `cube` est de classe `Cube`, et `roue_0` et `roue_1` sont de classe `Roue`.
    — Un constructeur qui place les deux roues sur le cube à l'aide des instructions suivantes : `roue_0.connecter(0,cube,2); roue_1.connecter(0,cube,6);`
    — Une fonction membre `rouler()` sans arguments, se contentant de tourner les deux roues d'un angle de `90` pendant `3` secondes à l'aide de la fonction `tournerPendant()`.
    On écrira séparément :

    a) Un fichier source de déclaration de la classe `Charrette`.

    b) Un fichier source de définition de la classe `Charrette`.

    c) Un programme d'essai (main) qui utilise la classe `Charrette`, pour créer une charrette et la faire rouler.

    .....

    .....

    .....

    .....

CODE A. Soit le fichier `RobotCode.hpp` suivant :

```
#pragma once
#include "Base.hpp"
#include "Ressort.hpp"
#include "Engrenage.hpp"
#include "BifurcationY.hpp"

class RobotCode{

   private:
      Base base;
      BifurcationY bifurcationy;
      Engrenage engrenage;
      Ressort ressort;

      void initialiser();

   public:

      RobotCode();
      void animer();
      void reinitialiser();
};
```

CODE B. Soit le fichier `RobotCode.cpp` suivant :

```
#include "RobotCode.hpp"

RobotCode::RobotCode(){

  ressort.connecter(0,base,18);
  engrenage.connecter(0,ressort,1);
  bifurcationy.connecter(0,engrenage,1);

  initialiser();
}

void RobotCode::initialiser(){

   engrenage.angleDeRotation = 45;
   engrenage.couleur = "#ffffff";
   base.couleur = "#ffffff";
   bifurcationy.couleur = "#00ff00";
   ressort.couleur = "#ff0000";
}

void RobotCode::reinitialiser(){
```

```
23     initialiser();
   }

   void RobotCode::animer(){

28     engrenage.tournerPendant(90,3);
       base.colorierPendant("#ff0000",10);
   }
```

CODE C. Soit le fichier `main.cpp` suivant :

```
c) Fichier main.cpp :

#include "RobotCode.hpp"
4
void main () {

   RobotCode robotCode;
   robotCode.animer();
9 }
```

F.3 QUESTIONNAIRE SUR LES APPRÉCIATIONS SUBJECTIVE D'AMUSEMENT ET DE DÉTERMINATION

Ces questions ont été posées suite à une séance expérimentale dédiée à l'utilisation d'objets (expérimentations réalisées avec PrOgO v.0.6 et v.0.7) :

1. Réaliser une construction 3D vous a-t-il été amusant ?
   - (A) pas du tout
   - (B) peu
   - (C) plutôt
   - (D) très
2. Durant la réalisation de votre construction 3D, avez vous désiré :
   - (A) aller jusqu'au bout
   - (B) vous arrêter avant la fin

Ces questions ont été posées suite à une séance expérimentale dédiée à la création de classes (expérimentation réalisée avec PrOgO v.0.7) :

1. Modifier une construction 3D vous a-t-il été amusant ?
   - (A) pas du tout
   - (B) peu
   - (C) plutôt
   - (D) très

2. Durant la modification de votre construction 3D, avez vous désiré :
   (A) aller jusqu'au bout
   (B) vous arrêter avant la fin

F.4 QUESTIONNAIRE SUR LES AVIS DES PARTICIPATIONS UTILISÉ AVEC LE PROTOTYPE EXPÉRIMENTAL DE PROGO

1. Quel âge avez-vous ?
   .....
2. Jouez-vous aux jeux vidéos ?
   (A) Oui
   (B) Non

   Si Oui, jouez-vous :

   (A) Rarement
   (B) souvent
   (C) Fréquemment
   (D) Très fréquemment
3. Jouez-vous aux jeux sérieux ?
   (A) Oui
   (B) Non

   Si Oui, jouez-vous :

   (A) Rarement
   (B) souvent
   (C) Fréquemment
   (D) Très fréquemment
4. Avez-vous trouvé le jeu amusant ?
   (A) pas du tout
   (B) peu
   (C) plutôt
   (D) beaucoup
5. Avez-vous trouvé le jeu pédagogique ?
   (A) pas du tout
   (B) peu
   (C) plutôt
   (D) beaucoup

6. Avez-vous trouvé le jeu attractif ?
   (A) pas du tout
   (B) peu
   (C) plutôt
   (D) beaucoup
7. Avez-vous eu le sentiment d'avoir appris ?
   (A) Oui
   (B) Non

   Si Oui, pourriez-vous résumer en quelques lignes ce que vous avez appris ?

   .....

   .....
8. Souhaiteriez-vous une future utilisation de PrOgO, afin d'apprendre d'autres concepts ?
   (A) Oui
   (B) Non
9. Quels aspects du jeu voudriez-vous voire s'améliorer ?

   .....

G

MÉTHODES ET PARAMÈTRES D'ANALYSE STATISTIQUE

G.1 ANALYSE EN COMPOSANTES PRINCIPALES (ACP)

G.1.1 *Principes et objectifs*

L'idée centrale dans l'ACP [1] est de réduire la *dimensionnalité* de données contenant un grand nombre de variables fortement corrélées, tout en retenant autant que possible la *variabilité* des données. Cette réduction est réalisée en transformant les variables initiales en un nouvel ensemble de variables non corrélées, *les composantes principales*, appelées aussi *facteurs* ou *axes*. Ces dernières sont ordonnées de sorte à ce que les premières retiennent le maximum de variabilité de l'ensemble des variables initiales (Jolliffe, 2002).

Cette réduction possède un avantage, notamment, dans le cas où les données ont plusieurs variables qui sont en réalité proches de deux axes particuliers formant un plan. Par conséquent, les données peuvent être projetées sur ce plan, offrant une représentation visuelle des données plus faciles à interpréter que si elles sont présentées en forme d'une masse de chiffres.

G.1.2 *Résultats de l'ACP et leur interprétation*

Un des premiers résultats retournés par l'ACP consiste en les composantes principales (facteurs ou axes) retournées avec leurs valeurs propres, reflétant la variabilité des variables initiales venant d'être réduites. Idéalement, un nombre restreint de facteurs ayant des valeurs propres élevées sont retenues, afin d'assurer une bonne représentation visuelle des données (Jolliffe, 2002).

Un second résultat important de l'ACP est le *cercle des corrélations*, qui montre une projection des variables initiales dans un sous-espace bidimensionnel (un plan) contenant deux axes qui sont de préférence les deux premiers (ayant des valeurs propres maximales). La corrélation reflète le degré de dépendance entre deux variables. Dans notre cas, elle est mesurée par le *coefficient de Pearson* ayant une valeur comprise entre -1 et +1. Plus le coefficient est proche de -1 ou de +1, plus la corrélation entre les variables est bonne. Afin d'assurer une bonne interprétation du sens des axes, les variables doivent être placées loin du centre du cercle de corrélation. Une variable est fortement liée à un axe, quand sa valeur absolue sur cet axe est élevée (proche de 1). Les variables qui contribuent à la définition d'un axe sont celles qui maximisent leurs valeurs absolues sur cet axe.

1. Voire l'ouvrage de Lebart, Morineau et Piron (1995) pour les détails des calculs sur lesquels l'ACP est fondée.

Un autre résultat important de l'ACP est le *graphe des observations*, qui permet de représenter les observations sur un plan constitué de préférence par les deux premiers axes (ayant des valeurs propres maximales). L'interprétation du *graphe des observations* doit se faire après avoir dégagé du *cercle des corrélations* la signification des axes. Cela permet de déduire les tendances de l'ensemble des individus (observations).

G.2 Classification ascendante hiérarchique (CAH)

L'objectif de la méthode de la CAH [2] est de construire des partitions emboîtées d'un ensemble d'individus à partir de leurs distances deux à deux. Cet algorithme part de la partition constituée par tous les éléments séparés et, à chaque étape, il réunit les deux sous-ensembles « les plus proches » pour constituer un nouveau sous-ensemble, jusqu'à l'obtention de l'ensemble global. Cette façon de procéder est représentée par un arbre de classification (*dengrogramme*) (Lebart, Morineau et Piron, 1995). Il représente de manière claire la façon dont l'algorithme procède pour regrouper les individus puis les sous-groupes. Il montre comment l'algorithme regroupe progressivement toutes les observations.

L'intérêt de cet arbre est qu'il peut donner une idée du nombre de classes existant effectivement dans la population. Chaque coupure d'un arbre fournit une partition, ayant d'autant moins de classes et des classes d'autant moins homogènes que l'on coupe plus haut (Lebart, Morineau et Piron, 1995). Grâce à la troncature (ligne en pointillé traversant le dendrogramme), on peut visualiser que des groupes homogènes ont été formés.

G.3 Paramètres statistiques utilisés dans l'analyse des scores obtenus aux tests de connaissances

Afin d'analyser de manière significative les scores obtenus à un test de vérification de connaissances et d'estimer la qualité de ce dernier, trois paramètres statistiques sont calculés, comme suggéré par Yuan et al. (2013), à savoir un *indice de difficulté*, un *indice de discrimination* et un *indice de fiabilité*.

G.3.1 *Indice de difficulté*

L'indice de difficulté peut être calculé au niveau de la question comme au niveau du test dans son ensemble. Il est défini comme la proportion d'étudiants répondant correctement au test (Yuan et al., 2013). Étant donné que les tests élaborés pour nos expérimentations contiennent des questions de différents types et sont notées différemment, la note globale du test n'est pas calculée en fonction des réponses correctes ou non correctes, mais en fonction des notes normalisées des différentes questions. Ces dernières possèdent le même niveau d'importance (notes normalisée à 1). Nous avons donc procédé en calculant un indice de difficulté pour chacun des trois premiers niveaux de la taxonomie, en plus de l'indice de difficulté globale (calculé au niveau du test dans

2. Voire l'ouvrage de Lebart, Morineau et Piron (1995) pour les détails de l'algorithme CAH.

son ensemble). Un indice de difficulté est désigné par la lettre P, il est équivalent à la moyenne du test et calculé comme suit :

$$P = \frac{1}{n} \sum_{i=1}^{n} A_i \tag{1}$$

Où : A_i est le score moyen obtenu à un groupe de questions pour le participant i, n est le nombre de participants.

Plus les étudiants répondent correctement à un ensemble de questions, moins est la difficulté des questions. Idéalement, il convient d'avoir la valeur de l'indice de difficulté P comprise entre 0.3 et 0.8. Dans le cas où P est supérieur à 0.8, le test est considéré *facile*. Il est considéré plutôt difficile quand P est inférieur à 0.3 (McDonald, 2013).

G.3.2 *Indice de discrimination*

L'indice de discrimination sert à estimer combien le test a permis de discriminer les étudiants ayant obtenu des scores élevés de ceux ayant obtenu des scores faibles (Kelley, 1939 ; Yuan et al., 2013). Il est défini comme la différence entre la moyenne des 27% des meilleurs scores et la moyenne des 27% des scores les plus faibles. De la même manière que pour l'indice de difficulté, un indice de discrimination pour chacun des trois premiers niveaux de la taxonomie a été calculé. En plus de l'indice de discrimination global (calculé au niveau du test dans son ensemble). Un indice de discrimination est désigné par la lettre D et calculé comme suit :

$$D = A_e - A_f \tag{2}$$

Où : A_e est la moyenne des 27% des scores moyens les plus élevés, A_f est la moyenne des 27% des scores moyens les plus faibles.

Idéalement, il convient d'avoir la valeur de l'indice de discrimination D supérieure à 0.39 ($D > 0.39$). Cela indique que le test discrimine très bien les étudiants ayant les scores les plus élevés des étudiants ayant les scores les plus faibles. Quand D est compris entre 0.3 et 0.39 ou entre 0.2 et 0.29 ($0.2 < D < 0.29$), le test est qualifié de *passable* et nécessite d'être amélioré. Le test doit être rejeté, quand D est inférieur à 0.2 ($D < 0.2$) (Kelley, 1939).

G.3.3 *Indice de fiabilité*

L'indice de fiabilité donne la probabilité d'obtention de résultats similaires quand le même test est utilisé auprès d'autres étudiants similaires. Une telle mesure peut être donnée par le *coefficient alpha de Cronbach*, comme suggéré par (Yuan et al., 2013). Le coefficient alpha de Cronbach est désigné par la lettre α, il est calculé par la formule suivante :

$$\alpha = \frac{k}{k-1}\left(1 - \frac{\sum_{i=1}^{k} \sigma_i^2}{\sigma_x^2}\right) \tag{3}$$

Où : k est le nombre d'items, σ_i^2 est la variance des scores de l'item i et σ_x^2 est la variance des scores globaux.

Dans notre cas, la variance est calculée non pas pour chaque question, mais pour chacun des trois premiers niveaux de la taxonomie.

Il convient d'avoir une valeur comprise entre 0.6 et 0.8 pour dire que le test est *fiable* ou *consistant* (Yuan et al., 2013).

BIBLIOGRAPHIE

Abiteboul, Serge, Jean-Pierre Archambaul, Christine Balagué, Georges-Louis Baron, Gérard Berry, Gilles Dowek et Colin De la Higuera (2013). *Rapport de l'académie des sciences. L'enseignement de l'informatique en France. Il est urgent de ne plus attendre*. Institut de France. Académie des sciences. (cf. p. 2).

Adams, Joel et Jeremy Frens (2003). « Object centered design for Java : teaching OOD in CS-1 ». In : *ACM SIGCSE Bulletin*. T. 35. 1. ACM, p. 273–277 (cf. p. 47, 49, 50).

AlAgha, Iyad, Andrew Hatch, Linxiao Ma et Liz Burd (2010). « Towards a teacher-centric approach for multi-touch surfaces in classrooms ». In : *ACM International Conference on Interactive Tabletops and Surfaces*. ACM, p. 187–196 (cf. p. 93).

Alvarez, Julian (2007). « Du jeu vidéo au serious game ». Thèse de doct. Université Toulouse (cf. p. 10).

Amory, Alan (2007). « Game object model version II : a theoretical framework for educational game development ». In : *Educational Technology Research and Development* 55.1, p. 51–77 (cf. p. 12, 27, 29, 31).

Anderson, Lorin W et David R Krathwohl (2001). *A taxonomy for learning, teaching, and assessing : A revision of Bloom's Taxonomy of Educational Objectives*. Sous la dir. de New York : Addison Wesley Longmann (cf. p. 143, 191).

Anneta, Leonard, Richard Lamb et Marcus Stone (2011). « Assessing Serious Educational Games ». In : *Serious Educational Game Assessment*. Springer, p. 75–93 (cf. p. 9, 18, 27–31).

Annetta, Leonard A, James Minogue, Shawn Y Holmes et Meng-Tzu Cheng (2009). « Investigating the impact of video games on high school students' engagement and learning about genetics ». In : *Computers & Education* 53.1, p. 74–85 (cf. p. 25).

Baldwin, Lynne P et Jasna Kuljis (2001). « Learning programming using program visualization techniques ». In : *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on*. IEEE, 8–pp (cf. p. 84).

Barbel, Patrice (2003). « Liens entre utilisabilité et utilité d'un logiciel éducatif en situation d'enseignement ». In : *Environnements Informatiques pour l'Apprentissage Humain*, p. 405–412 (cf. p. 21).

Barber, Michael (1997). *The learning game : Arguments for an education revolution*. Indigo (cf. p. 11).

Barbier, Franck (2009). *Conception orientée objet en Java et C++*. Pearson Education France (cf. p. 38, 39, 109).

Bashiru, Lawal et Agunlejika Aderogba Joseph (2015). « Learning difficulties of Object Oriented Programming (OOP) in University of Ilorin - Nigeria : Students perspectives ». In : *Eighth TheIIER-Science Plus International Conference* (cf. p. 45, 46, 50).

Baum, Dave et Rodd Zurcher (2003). *Definitive guide to Lego Mindstorms*. T. 2. Apress (cf. p. 86).

Becker, Byron Weber (2001). « Teaching CS1 with karel the robot in Java ». In : *ACM SIGCSE Bulletin*. T. 33. 1. ACM, p. 50–54 (cf. p. 35, 47, 48, 50).

Begel, Andrew (1996). « LogoBlocks : A graphical programming language for interacting with the world ». In : *Electrical Engineering and Computer Science Department, MIT, Boston, MA* (cf. p. 62, 64, 65, 86).

Bennedsen, Jens (2008). « Teaching and learning introductory programming : a model-based approach ». Thèse de doct. University of Oslo. Faculty of Mathematics & Natural Sciences (cf. p. 35, 47, 82).

Bennedsen, Jens, Michael E Caspersen et Michael Kölling (2008). *Reflections on the teaching of programming : methods and implementations*. T. 4821. Springer (cf. p. 35).

Bennedsen, Jens et Carsten Schulte (2007). « What does Objects-First mean ? : An international study of teachers' perceptions of Objects-First ». In : *seventh Baltic Sea Conference on Computing Education Research*. T. 88. Australian Computer Society, p. 21–29 (cf. p. 35, 47, 50, 175).

Bergin, Joseph, Jim Roberts, Richard Pattis et Mark Stehlik (1997). *Karel++ : A gentle introduction to the art of Object-Oriented Programming*. John Wiley & Sons, Inc. (cf. p. 64, 69, 70, 86).

Bergin, Joseph, Mark Stehlik, Jim Roberts et Rich Pattis (2005). *Karel J Robot : A gentle introduction to the art of object-oriented programming in Java*. Dream Songs Press (cf. p. 69, 71).

Biehl, Jacob T et Brian P Bailey (2004). « ARIS : an interface for application relocation in an interactive space ». In : *Proceedings of Graphics Interface*. Canadian Human-Computer Communications Society, p. 107–116 (cf. p. 93).

Biju, Soly Mathew (2013). « Difficulties in understanding object oriented programming concepts ». In : *Innovations and Advances in Computer, Information, Systems Sciences, and Engineering*. Springer, p. 319–326 (cf. p. 45, 46, 50).

Black, Andrew P (2013). « Object-Oriented programming : Some history, and challenges for the next fifty years ». In : *Information and Computation* 231, p. 3–20 (cf. p. 44).

Bliss, Joan et Jon Ogborn (1989). « Tools for exploratory learning : A research programme* ». In : *Journal of Computer Assisted Learning* 5.1, p. 37–50 (cf. p. 85).

Brown, Marc H et Robert Sedgewick (1984). « A system for algorithm animation ». In : *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. T. 18. 3. ACM, p. 177–186 (cf. p. 63).

Brown, Randy (2011). « Assessment using after-action review ». In : *Serious Educational Game Assessment*. Springer, p. 119–129 (cf. p. 23).

Bruce, Kim B (2005). « Controversy on how to teach CS 1 : a discussion on the SIGCSE-members mailing list ». In : *ACM SIGCSE Bulletin* 37.2, p. 111–117 (cf. p. 35).

Bruillard, Eric (1997). *Les machines à enseigner*. Hermès (cf. p. 77–79, 81, 87).

Buck, Duane et David J Stucki (2000). « Design early considered harmful : Graduated exposure to complexity and structure based on levels of cognitive development ». In : *ACM SIGCSE Bulletin*. T. 32. 1. ACM, p. 75–79 (cf. p. 47, 48).

— (2001). « JKarelRobot : a case study in supporting levels of cognitive development in the computer science curriculum ». In : *ACM SIGCSE Bulletin* 33.1, p. 16–20 (cf. p. 72).

Burd, Liz, John Adams, Iyad AlAgha, Tom Barrett, Joe Julian Elliott, Andrew Hatch, Steve Higgins et al. (2013). *SynergyNet. Mutli-touch in education*. URL : https://community.dur.ac.uk/tel.lab/synergynet/ (cf. p. 93).

Caillois, Roger (1991). *Les jeux et les hommes : le masque et le vertige*. T. 184. Editions Gallimard (cf. p. 9).

Carroll, John M et Robert L Mack (1999). « Metaphor, computing systems, and active learning ». In : *International Journal of Human-Computer Studies* 51.2, p. 385–403 (cf. p. 83, 85, 89).

Carroll, John M, Mary Beth Rosson et Mark K Singley (1993). « The collaboration thread : A formative evaluation of Object-Oriented education ». In : *Empirical Studies of Programmers : Fifth Workshop*. Palo Alto, CA, p. 26–41 (cf. p. 45).

Christophe, Dabancourt (2005). *Apprendre à programmer, algorithmes et conception objet*. Eyrolles (cf. p. 36).

Clarisse, Olivier et Shi-Kuo Chang (1986). « Vicon : A Visual icon manager ». In : *Visual languages*. Springer, p. 151–190 (cf. p. 63, 86).

Collazos, César A, Luis A Guerrero, José A Pino et Sergio F Ochoa (2002). « Evaluating collaborative learning processes ». In : *Groupware : Design, Implementation, and Use*. Springer, p. 203–221 (cf. p. 24).

Conati, Cristina et Christina Merten (2007). « Eye-tracking for user modeling in exploratory learning environments : An empirical evaluation ». In : *Knowledge-Based Systems* 20.6, p. 557–574 (cf. p. 172).

Conway, Matthew J (1997). « Alice : easy-to-learn 3D scripting for novices ». Thèse de doct. University of Virginia (cf. p. 72, 81).

Conway, Matthew, Steve Audia, Tommy Burnette, Dennis Cosgrove et Kevin Christiansen (2000). « Alice : lessons learned from building a 3D system for novices ». In : *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, p. 486–493 (cf. p. 64, 72).

Cooper, Stephen, Wanda Dann et Randy Pausch (2000). « Alice : a 3-D tool for introductory programming concepts ». In : *Journal of Computing Sciences in Colleges*. T. 15. 5. Consortium for Computing Sciences in Colleges, p. 107–116 (cf. p. 3, 54, 72, 81, 86, 89).

— (2003). « Teaching objects-first in introductory computer science ». In : *ACM SIGCSE Bulletin*. T. 35. 1. ACM, p. 191–195 (cf. p. 4, 47, 49, 50, 82, 84, 85).

Courtin, Jacques et Irène Kowarski (1994). *Initiation à l'algorithmique et aux structures de données*. Dunod (cf. p. 36).

Cunniff, Nancy, Robert P Taylor et John B Black (1986). « Does programming language affect the type of conceptual bugs in beginners' programs ? A comparison of FPL and Pascal ». In : *ACM SIGCHI Bulletin* 17.4, p. 175–182 (cf. p. 63).

D. Foley, James, Andries van Dam, Steven K. Feiner et John F. Hughes (1995). *Computer graphics principles and practice*. Addison-Wesley (cf. p. 124).

Dann, Wanda, Dennis Cosgrove, Don Slater, Dave Culyba et Steve Cooper (2012). « Mediated transfer : Alice 3 to java ». In : *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, p. 141–146 (cf. p. 4, 74).

Dawson, Shane, Dragan Gašević, George Siemens et Srecko Joksimovic (2014). « Current state and future trends : A citation network analysis of the learning analytics

field ». In : *Proceedings of the Fourth International Conference on Learning Analytics And Knowledge*. ACM, p. 231–240 (cf. p. 24).

Day, William HE et Herbert Edelsbrunner (1984). « Efficient algorithms for agglomerative hierarchical clustering methods ». In : *Journal of classification* 1.1, p. 7–24 (cf. p. 143, 145).

De Freitas, Sara et Martin Oliver (2006). « How can exploratory learning with games and simulations within the curriculum be most effectively evaluated ? » In : *Computers & Education* 46.3, p. 249–264 (cf. p. 9, 12, 22, 27, 29–31).

DeMichiel, Linda G et Richard P Gabriel (1987). « The common lisp object system : An overview ». In : *European Conference on Object-Oriented Programming*. Springer, p. 151–170 (cf. p. 43, 44).

Di Loreto, Ines et Abdelkader Gouaich (2010). « An early evaluation method for social presence in serious games ». In : *2nd International Conference on Computer Supported Education* (cf. p. 22, 29, 31).

Djelil, Fahima (2014). « Comment évaluer un jeu d'apprentissage en contexte de formation professionnelle ». In : *Rencontres de Jeunes Chercheurs en EIAH*, p. 11–16 (cf. p. 22, 31).

Djelil, Fahima, Adélaïde Albouy-Kissi, Benjamen Albouy-kissi, Eric Sanchez et Jean-Marc Lavest (2016). « Microworlds for learning Object-Oriented Programming : Considerations from research to practice ». In : *Joural of Interactive Learning Research* 27.3, p. 265–284 (cf. p. 82, 84).

Djelil, Fahima, Benjamin Albouy-Kissi, Adélaïde Albouy-Kissi, Eric Sanchez et Jean-Marc Lavest (2015). « Towards a 3D virtual game for learning Object-Oriented Programming fundamentals and C++ language - Theoretical considerations and empirical results ». In : *Proceedings of the 7th International Conference on Computer Supported Education*, p. 289–294 (cf. p. 137).

Djelil, Fahima, Eric Sanchez, Benjamin Albouy-Kissi, Jean-Marc Lavest et Adélaïde Albouy-Kissi (2014). « Towards a learning game evaluation methodology in a training context : A literature review ». In : *European Conference on Games Based Learning*. T. 2. Academic Conferences International Limited, p. 676–682 (cf. p. 22, 31).

Dondlinger, Mary Jo (2007). « Educational video game design : A review of the literature ». In : *Journal of Applied Educational Technology* 4.1, p. 21–31 (cf. p. 28–31, 81).

Dunleavy, Matt et Brittney Simmons (2011). « Assessing learning and identity in augmented reality science games ». In : *Serious Educational Game Assessment*. Springer, p. 221–240 (cf. p. 22).

Ellis, TO, John F Heafner et WL Sibley (1969). *The GRAIL Project : An experiment in man-machine communications*. Rapp. tech. DTIC Document (cf. p. 63).

Engel, Gerald et Eric Roberts (2001). *Computing curricula 2001. Computer science*. The Joint Task Force on Computing Curricula. (Cf. p. 39).

Esteves, Micaela et AJ Mendes (2003). « OOP-Anim, a system to support learning of basic object oriented programming concepts ». In : *Proceedings of CompSysTech'2003-International Conference on Computer Systems and Technologies. Sofia, Bulgaria* (cf. p. 84).

Etchegoyen, Alain (2004). *La métaphore de la construction*. Constructif N°7 : se former tout au long de sa vie. url : http://www.constructif.fr/bibliotheque/2004-1/la-metaphore-de-la-construction.html?item_id=2527 (cf. p. 108).

Everitt, Katherine, Chia Shen, Kathy Ryall et Clifton Forlines (2006). « MultiSpace : Enabling electronic document micro-mobility in table-centric, multi-device environments ». In : *First IEEE International Workshop on Horizontal Interactive Human-Computer Systems*. IEEE, 8–pp (cf. p. 93).

Garris, Rosemary, Robert Ahlers et James E Driskell (2002). « Games, motivation, and learning : A research and practice model ». In : *Simulation & gaming* 33.4, p. 441–467 (cf. p. 9, 12, 13, 15, 27–31, 81).

Genvo, Sébastien (2011). « Penser les phénomènes de ludicisation du numérique : pour une théorie de la jouabilité ». In : *Revue des sciences sociales* 45, p. 68–77 (cf. p. 11).

— (2012). « La théorie de la ludicisation : une approche anti-essentialiste des phénomènes ludiques ». In : *Journée d'études Jeu et jouabilité à l'ère numérique* (cf. p. 11).

George, Sébastien, Désirée Titon et Patrick Prévot (2005). « Simulateur de comportements d'apprenants dans le cadre de jeux d'entreprise ». In : *Environnements Informatiques pour l'Apprentissage Humain* (cf. p. 23).

Gilbert, John K (2010). « The role of visual representations in the learning and teaching of science : An introduction ». In : *Asia-Pacific Forum on Science Learning and Teaching*. T. 11. 1, p. 1–19 (cf. p. 84).

González-González, Carina, Pedro Toledo-Delgado, Cesar Collazos-Ordoñez et José L González-Sánchez (2013). « Design and analysis of collaborative interactions in social educational videogames ». In : *Computers in Human Behavior* (cf. p. 25).

Granet, Vincent (2014). *Algorithmique et programmation en Java-4e éd. : Cours et exercices corrigés*. Dunod (cf. p. 36, 37).

Guyot, Wally M (1978). « Summative and formative evaluation ». In : *The Journal of Business Education* 54.3, p. 127–129 (cf. p. 23, 24).

Hadjerrouit, Said (2005). « Object-Oriented software development education : a constructivist framework. » In : *Informatics in Education* 4.2, p. 167–192 (cf. p. 85, 89).

Harel, Idit et Seymour Papert (1991). *Constructionism.* Ablex Publishing (cf. p. 81).

Henriksen, Poul et Michael Kölling (2004). « Greenfoot : combining object visualisation with interaction ». In : *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. ACM, p. 73–82 (cf. p. 82, 84).

Henriot, Jacques (1969). *Le jeu*. Presses Universitaire de France (cf. p. 9, 10, 27).

Hogle, Jan G (1995). « Computer Microworlds in Education : Catching Up with Danny Dunn. » In : (cf. p. 77–79, 85).

Hoyles, Celia, Richard Noss et Ross Adamson (2002). « Rethinking the microworld idea ». In : *Journal of educational computing research* 27.1, p. 29–53 (cf. p. 84).

Hudak, Paul (1989). « Conception, evolution, and application of functional programming languages ». In : *ACM Computing Surveys (CSUR)* 21.3, p. 359–411 (cf. p. 37).

Huizinga, Johan (1951). « Homo ludens. Essai sur la fonction sociale du jeu ». In : *Paris : Editions Gallimard* (cf. p. 9).

Jerding, Dean Frederick et John T Stasko (1994). *Using visualization to foster object-oriented program understanding*. Rapp. tech. Georgia Institute of Technology (cf. p. 84).

Jolliffe, Ian (2002). *Principal component analysis*. Springer (cf. p. 143, 202).

Julien, Charpentier et Cyrille Grandval (2015). *PHP. Préparation à la certification Zend Certified PHP Engineer (ZCPE)*. Editions ENI (cf. p. 131).

Kafai, Yasmin B (2006). « Playing and making games for learning instructionist and constructionist perspectives for game studies ». In : *Games and culture* 1.1, p. 36–40 (cf. p. 9, 84).

Kaltenbrunner, Martin (2009). « reacTIVision and TUIO : A tangible tabletop toolkit ». In : *Proceedings of the ACM international Conference on interactive Tabletops and Surfaces*. ACM, p. 9–16 (cf. p. 96, 97).

Kaltenbrunner, Martin, Till Bovermann, Ross Bencina et Enrico Costanza (2005). « TUIO : A protocol for table-top tangible user interfaces ». In : *6th International Workshop on Gesture in Human-Computer Interaction and Simulation*, p. 1–5 (cf. p. 95, 97).

Kay, Alan C (1996). « The early history of Smalltalk ». In : *History of programming languages—II*. ACM, p. 511–598 (cf. p. 43).

Kelleher, Caitlin, Randy Pausch et Sara Kiesler (2007). « Storytelling alice motivates middle school girls to learn computer programming ». In : *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, p. 1455–1464 (cf. p. 84, 85).

Kelley, Truman L (1939). « The selection of upper and lower groups for the validation of test items ». In : *Journal of Educational Psychology* 30.1, p. 17 (cf. p. 204).

Kiili, Kristian (2005). « Digital game-based learning : Towards an experiential gaming model ». In : *The Internet and higher education* 8.1, p. 13–24 (cf. p. 9, 15, 16, 28–31).

Klein, James D et Eric Freitag (1991). « Effects of using an instructional game on motivation and performance ». In : *The Journal of Educational Research* 84.5, p. 303–308 (cf. p. 11).

Koenig, Alan D, John Lee, Markus Iseli et Richard Wainess (2009). « A conceptual framework for assessing performance in games and simulations ». In : *The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC)*. T. 2009. 1. NTSA (cf. p. 26).

Kölling, Michael (2003). « The curse of hello world ». Oslo, Norway : Invited lecture at Workshop on Learning and Teaching Object-Orientation – Scandinavian Perspectives (cf. p. 45–47, 50).

— (2010a). *Introduction to programming with Greenfoot*. Pearson Education, Upper Saddle River, New Jersey, USA (cf. p. 48, 50, 64, 74, 84).

— (2010b). « The greenfoot programming environment ». In : *ACM Transactions on Computing Education (TOCE)* 10.4, p. 14 (cf. p. 3, 54, 74, 83, 84, 86).

Kölling, Michael et John Rosenberg (2001). « Guidelines for teaching object orientation with Java ». In : *ACM SIGCSE Bulletin*. T. 33. 3. ACM, p. 33–36 (cf. p. 35, 47, 48).

— (2002). *BlueJ-The hitch-hikers guide to Object-Orientation*. Rapp. tech. The Maersk McKinney Moller Institue for Production Technology University of Southern Denmark, p. 1–8 (cf. p. 45, 47, 50).

Lakoff, George et Mark Johnson (2008). *Metaphors we live by*. University of Chicago press (cf. p. 83).

Lapedes, Daniel N (2002). *McGraw-Hill dictionary of scientific and technical terms*. McGraw-Hill Book Co., New York, NY (cf. p. 95).

Lawler, Robert W (1979). « One child's learning : introducing writing with a computer ». Thèse de doct. Massachusetts Institute of Technology - Artificial Intelligence Laboratory (cf. p. 67).
— (1987). « Artificial Intelligence and Education : Learning environments and tutoring systems ». In : t. 1. Intellect Books. Chap. Learning Environments. Now, Then and Someday (cf. p. 77, 82, 85, 175).
Lawler, Robert W et Masoud Yazdani (1987). *Artificial Intelligence and Education : Learning environments and tutoring systems*. T. 1. Intellect Books (cf. p. 77).
Lebart, Ludovic, Alain Morineau et Marie Piron (1995). *Statistique exploratoire multidimensionnelle*. Dunod (cf. p. 202, 203).
Loh, Christian S (2012). « Information trails : In-process assessment of game-based learning ». In : *Assessment in Game-Based Learning*. Springer, p. 123–144 (cf. p. 23).
MacConnell, Steve (1993). *Code complete : a practical handbook of software construction*. Microsoft Press (cf. p. 83).
Malone, Thomas (1981). *What makes computer games fun ?* T. 13. 2-3. ACM (cf. p. 81).
Manin, Nicolas, Sébastien George et Patrick Prévot (2006). « Using virtual learners' behaviours to help the development of educational business games ». In : *Innovative Approaches for Learning and Knowledge Sharing*. Springer, p. 287–301 (cf. p. 23).
Marfisi, Iza, Sébastien George, Franck Tarpin-Bernard et Patrick Prévot (2012). « Comment évaluer la qualité d'un Learning Game pendant sa conception ? » In : *Technologies de l'information et de la communication pour l'enseignement*, p. 80–90 (cf. p. 19, 22, 27–31).
Marfisi-Schottman, Iza (2012). « Méthodologie, modèles et outils pour la conception de Learning Games ». Thèse de doct. Institut National des Sciences Appliquées de Lyon (cf. p. 11).
Marne, Bertrand, Benjamin Huynh-Kim-Bang et Jean-Marc Labat (2011). « Articuler motivation et apprentissage grâce aux facettes du jeu sérieux ». In : *Actes de la conférence EIAH*, p. 69–80 (cf. p. 22).
Martin, Fred, Bakhtiar Mikhak, Michel Resnick, Brian Silverman et Robbie Berg (2000). « To mindstorms and beyond. Evolution of a contruction kit for magical machines ». In : Morgan Kaufman Publishers (cf. p. 56, 64).
Martinez-Maldonado, Roberto, Yannis Dimitriadis, Judy Kay, Kalina Yacef et Marie-Theresa Edbauer (2013). « MTClassroom and MTDashboard : supporting analysis of teacher attention in an orchestrated multi-tabletop classroom ». In : *International Conference on Computer supported collaborative Learning*, p. 119–128 (cf. p. 93).
Mayer, Richard E (1975). « Different problem-solving competencies established in learning computer programming with and without meaningful models. » In : *Journal of Educational Psychology* 67.6, p. 725 (cf. p. 83).
— (1981). « The psychology of how novices learn computer programming ». In : *ACM Computing Surveys (CSUR)* 13.1, p. 121–141 (cf. p. 83, 89).
McDonald, Mary E (2013). *The nurse educator's guide to assessing learning outcomes*. Jones & Bartlett Publishers (cf. p. 204).
Mcgettrick, Andrew, Roger Boyle, Roland Ibbett, John Lloyd, Gillian Lovegrove et Keith Mander (2005). « Grand challenges in computing : Education—a summary ». In : *The Computer Journal* 48.1, p. 42–48 (cf. p. 35).

McNerney, Timothy Scott (1999). « Tangible programming bricks : An approach to making programming accessible to everyone ». Mém.de mast. Massachusetts Institute of Technology (cf. p. 56).
— (2004). « From turtles to Tangible Programming Bricks : explorations in physical language design ». In : *Personal and Ubiquitous Computing* 8.5, p. 326–337 (cf. p. 54, 56, 60).
Mettier, Yves (2014). *C en action (3ième édition)*. Editions ENI (cf. p. 131).
Milne, Iain et Glenn Rowe (2002). « Difficulties in learning and teaching programming—views of students and tutors ». In : *Education and Information technologies* 7.1, p. 55–66 (cf. p. 45, 50).
— (2004). « Ogre : Three-dimensional program visualization for novice programmers ». In : *Education and Information Technologies* 9.3, p. 219–237 (cf. p. 84).
Minsky, Marvin et Seymour Papert (1972). *The 72' Progress Report* (cf. p. 77).
Mislevy, Robert J et Michelle M Riconscente (2005). « Evidence-centered assessment design : Layers, structures, and terminology ». In : *Menlo Park, CA : SRI International* (cf. p. 26).
Moreno, Andrés, Niko Myller, Erkki Sutinen et Mordechai Ben-Ari (2004). « Visualizing programs with Jeliot 3 ». In : *Proceedings of the working conference on Advanced visual interfaces*. ACM, p. 373–376 (cf. p. 84).
Moreno-Ger, Pablo, Daniel Burgos, Iván Martínez-Ortiz, José Luis Sierra et Baltasar Fernández-Manjón (2008). « Educational game design for online education ». In : *Computers in Human Behavior* 24.6, p. 2530–2540 (cf. p. 11).
Moskal, Barbara, Deborah Lurie et Stephen Cooper (2004). « Evaluating the effectiveness of a new instructional approach ». In : *ACM SIGCSE Bulletin* 36.1, p. 75–79 (cf. p. 49, 54, 84).
Muratet, Mathieu (2010). « Conception, réalisation et évaluation d'un jeu sérieux de stratégie temps réel pour l'apprentissage des fondamentaux de la programmation ». Thèse de doct. Université Toulouse III-Paul Sabatier (cf. p. 64, 67, 68, 86).
Myers, Brad A (1990). « Taxonomies of visual programming and program visualization ». In : *Journal of Visual Languages & Computing* 1.1, p. 97–123 (cf. p. 62, 63).
Myers, Brad A, Ravinder Chandhok et Atul Sareen (1988). « Automatic data visualization for novice Pascal programmers ». In : *Proceedings of the IEEE*, p. 192–198 (cf. p. 63).
Naps, Thomas L et al. (2002). « Exploring the role of visualization and engagement in computer science education ». In : *ACM Sigcse Bulletin*. T. 35. 2. ACM, p. 131–152 (cf. p. 84, 89).
Natkin, Stéphane (2009). « Du ludo-éducatif aux jeux vidéo éducatifs ». In : *Les dossiers de l'ingénierie educative* 65, p. 12–15 (cf. p. 11).
Navarro, Emily Oh et André Van Der Hoek (2007). « Comprehensive evaluation of an educational software engineering simulation environment ». In : *20th Conference on Software Engineering Education & Training*. IEEE, p. 195–202 (cf. p. 23).
Nguyen, Dung X et Stephen B. Wong (2001). « OOP in introductory CS : Better students through abstraction ». In : *OOPSLA Fifth Workshop on Pedagogies and Tools for Assimilating Object-Oriented Concepts* (cf. p. 45, 50).

Nix, Don et Rand J Spiro (1990). *Cognition, education, and multimedia : Exploring ideas in high technology*. Routledge (cf. p. 85).

Nogry, Sandra, Stéphanie Jean-Daubias et Magali Ollagnier-Beldame (2004). « Évaluation des EIAH : une nécessaire diversité des méthodes ». In : *Technologies de l'Information et de la Connaissance dans l'Enseignement Supérieur et l'Industrie*, p. 265–271 (cf. p. 21, 22).

Oldham, Joseph D (2005). « What happens after Python in CS1 ? » In : *Journal of computing sciences in colleges* 20.6, p. 7–13 (cf. p. 35).

Osman, Waleed Ibrahim et Mudawi M Elmusharaf (2014). « Effectiveness of combining algorithm and program animation : A case study with data structure course ». In : *Issues in Informing Science and Information Technology* 11, p. 155–168 (cf. p. 84).

Oulhaci, Ali Mhammed, Erwan Tranvouez, Sébastien Fournier et Bernard Espinasse (2013). « Evaluation multi-critères et distribuée pour l'apprentissage collectif de procédures dans un jeux sérieux pour la gestion de crise ». In : *Actes de la Journée EIAH&IA* (cf. p. 23, 24).

Papastergiou, Marina (2009). « Digital Game-Based Learning in high school Computer Science education : Impact on educational effectiveness and student motivation ». In : *Computers & Education* 52.1, p. 1–12 (cf. p. 25).

Papert, Seymour (1980). *Mindstorms : Children, computers, and powerful ideas*. Basic Books (cf. p. 3, 43, 54, 55, 57, 58, 66, 77, 79, 81, 84–87, 89, 175).

Papert, Seymour (1987). « Artificial Intelligence and Education : Learning environments and tutoring systems ». In : t. 1. Intellect Books. Chap. Microworlds : Transforming education, p. 79–94 (cf. p. 77).

— (1993). *The children's machine : Rethinking school in the age of the computer*. Basic books (cf. p. 81).

Pears, Arnold, Stephen Seidman, Lauri Malmi, Linda Mannila, Elizabeth Adams, Jens Bennedsen, Marie Devlin et James Paterson (2007). « A survey of literature on the teaching of introductory programming ». In : *ACM SIGCSE Bulletin* 39.4, p. 204–223 (cf. p. 82).

Perkins, David N (1986). *Knowledge as design*. Routledge. Republié en 2013 (cf. p. 81).

Perlman, Radia (1976). *Using computer technology to provide a creative learning environment for preschool children*. Massachusetts Institute of Technology (cf. p. 54, 55, 59, 60, 81, 82, 86).

Pfahl, Dietmar, Oliver Laitenberger, Jörg Dorsch et Günther Ruhe (2003). « An externally replicated experiment for evaluating the learning effectiveness of using simulations in software project management education ». In : *Empirical software engineering* 8.4, p. 367–395 (cf. p. 25).

Piaget, Jean (1967). « Logique et connaissance scientifique ». In : *Encyclopédie de la Pléiade*. Gallimard. T. 22. Jean Piaget (cf. p. 77).

Plass, Jan L et Ruth N Schwartz (2014). « The Cambridge handbook of multimedia learning ». In : sous la dir. de Richard E. Mayer. 2e éd. Cambridge University Press. Chap. Multimedia learning with simulations and microworlds, p. 729–761 (cf. p. 84).

Prensky, Marc (2007). *Digital game-based learning*. T. 1. Paragon house St. Paul, MN (cf. p. 11).

Presser, Alan, Lee Farrell, Devon Kemp et W Lupton (2008). *Upnp device architecture 1.1*. Rapp. tech. UPnP Forum (cf. p. 97).
Raffle, Hayes Solos (2008). « Sculpting behavior : a tangible language for hands-on play and learning ». Thèse de doct. Massachusetts Institute of Technology (cf. p. 61, 62, 81, 82).
Raffle, Hayes Solos, Amanda J Parkes et Hiroshi Ishii (2004). « Topobo : a constructive assembly system with kinetic memory ». In : *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, p. 647–654 (cf. p. 54, 57, 61, 82, 86).
Reigeluth, Charles M et Ellen Schwartz (1989). « An instructional theory for the design of computer-based simulations. » In : *Journal of Computer-Based Instruction* 16.1, p. 1–10 (cf. p. 80).
Resnick, Mitchel, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan et all (2009). « Scratch : programming for all ». In : *Communications of the ACM* 52.11, p. 60–67 (cf. p. 3, 64–66, 86).
Rieber, Lloyd P. (1992). « Computer-based microworlds : A bridge between constructivism and direct instruction ». In : *Educational technology research and development* 40.1, p. 93–106 (cf. p. 85).
— (1994). *Computers, graphics & learning*. Brown & Benchmark Madison, WI (cf. p. 79, 80).
— (1996). « Seriously considering play : Designing interactive learning environments based on the blending of microworlds, simulations, and games ». In : *Educational technology research and development* 44.2, p. 43–58 (cf. p. 77–81).
— (2005). « The Cambridge handbook of multimedia learning ». In : sous la dir. de Richard E. Mayer. Cambridge University Press. Chap. Multimedia learning in games, simulations, and microworlds, p. 549–567 (cf. p. 79, 84).
Robins, Anthony, Janet Rountree et Nathan Rountree (2003). « Learning and teaching programming : A review and discussion ». In : *Computer science education* 13.2, p. 137–172 (cf. p. 35, 45).
Rosas, Ricardo et al. (2003). « Beyond Nintendo : Design and assessment of educational video games for first and second grade students ». In : *Computers & Education* 40.1, p. 71–94 (cf. p. 25).
Rupp, André A, Matthew Gushta, Robert J Mislevy et David Williamson Shaffer (2010). « Evidence-centered design of epistemic games : Measurement principles for complex learning environments ». In : *The Journal of Technology, Learning and Assessment* 8.4 (cf. p. 26).
Sajaniemi, Jorma, Pauli Byckling et Petri Gerdt (2007). « Animation metaphors for object-oriented concepts ». In : *Electronic Notes in Theoretical Computer Science* 178, p. 15–22 (cf. p. 84).
Sanchez, Eric (2011). *Key criteria for game design. A framework* (cf. p. 21).
— (2013). « A model for the design of digital epistemic games ». In : *World Conference on Computers in Education*, p. 257–264 (cf. p. 16, 17, 27, 29–31).
— (2014). *Le paradoxe du marionnettiste*. Thèse d'habilitation. Université Paris 5 Sorbonne Descartes (cf. p. 9–11).

Sanchez, Eric et Valérie Emin-Martinez (2014). « Towards a model of play : An empirical study ». In : *European Conference on Games Based Learning*. T. 2. Academic Conferences International Limited, p. 503–512 (cf. p. 25).

Sanchez, Eric, Valérie Emin-Martinez et Nadine Mandran (2015). « Jeu-game, jeu-play, vers une modélisation du jeu. Une étude empirique à partir des traces numériques d'interaction du jeu Tamagocours ». In : *Sciences et Technologies de l'Information et de la Communication pour l'Education et la Formation (STICEF)* 22 (cf. p. 9, 25, 27, 28, 33, 175).

Senach, Bernard (1990). « Evaluation ergonomique des Interfaces Homme-Machine : une revue de la littérature ». In : (cf. p. 21, 22).

Shaffer, David Williamson (2006). « Epistemic frames for epistemic games ». In : *Computers & education* 46.3, p. 223–234 (cf. p. 9, 11, 27, 33, 175).

Shaffer, David Williamson et al. (2009). « Epistemic network analysis : A prototype for 21st-century assessment of learning ». In : (cf. p. 26).

Sharp, Alec (1996). *Smalltalk by example : The developer's guide*. McGraw-Hill, Inc. (cf. p. 43).

Shneiderman, Ben (1981). « Direct manipulation : A step beyond programming languages ». In : *Proceedings of the Joint Conference on Easier and More Productive Use of Computer Systems*. T. 13. 2-3. ACM (cf. p. 63).

Shute, Valerie J, Lloyd Rieber et Richard Van Eck (2011). « Games... and... learning ». In : *Trends and issues in instructional design and technology* 3 (cf. p. 26).

Siemens, George et Phil Long (2011). « Penetrating the Fog : Analytics in learning and education. » In : *EDUCAUSE review* 46.5, p. 30 (cf. p. 24).

Sindre, Guttorm, Lasse Natvig et Magnus Jahre (2009). « Experimental validation of the learning effect for a pedagogical game on computer fundamentals ». In : *IEEE Transactions on Education* 52.1, p. 10–18 (cf. p. 11).

Sklenar, Jaroslav (1997). *Introduction to OOP in Simula*. url : http://staff.um.edu.mt/jskl1/talk.html#Basic_facts (cf. p. 42).

Smith, David Canfield (1975). *Pygmalion : a creative programming environment*. Rapp. tech. DTIC Document (cf. p. 63, 86).

Snyder, Alan (1991). « The essence of objects : Common concepts and terminology ». In : *Hewlett-Packard Company. Republié en 1993 dans IEEE Software* (cf. p. 44).

Stroustrup, Bjarne (1996). « History of programming languages-II ». In : sous la dir. de Thomas J. Bergin et Richard G. Gibson. ACM Press Books. Chap. A history of C++ : 1979–1991, p. 699–769 (cf. p. 43).

— (2013). *A tour of C++*. Addison Wesley (cf. p. 43).

Sturm, Oliver (2011). *Professional functional programming in C# : Classic programming techniques for modern projects*. John Wiley & Sons, Ltd (cf. p. 37).

Sutton-Smith, Brian (2009). *The ambiguity of play*. Harvard University Press (cf. p. 9).

Suzuki, H et H Kato (1993). « AlgoBlock : a tangible programming language, a tool for collaborative learning ». In : *Proceedings of 4th European Logo Conference*, p. 297–303 (cf. p. 54, 55).

Szilas, Nicolas et Denise Sutter Widmer (2013). « L'évaluation rapide de jeux d'apprentissage : la clef de voûte de l'ingénierie ludo-pédagogique (Instructional Game Design) ». In : EIAH, p. 24–28 (cf. p. 21).

Tapia, Andrea H, Magy Seif El-Nasr, Ibrahim Yucel, Joseph Zupko et Edgard Maldonado (2007). « Building virtual spaces. Games as gatekeepers for the IT workforce ». In : *Virtuality and Virtualization*. Springer, p. 317–334 (cf. p. 84).

Thomas, Pradeepa, Jean-Marc Labat, Mathieu Muratet et Amel Yessad (2012). « How to evaluate competencies in Game-Based Learning systems automatically ? » In : *Intelligent Tutoring Systems*. Springer, p. 168–173 (cf. p. 26).

Thomas, Pradeepa, Amel Yessad et Jean-Marc Labat (2011). « Réseaux de Petri et ontologies : des outils pour le suivi de l'apprenant dans les jeux sérieux ». In : *Actes de la conférence EIAH 2011*, p. 435–447 (cf. p. 26).

Thompson, Patrick.W (1988). « Artificial intelligence and instruction : Applications and methods ». In : JSTOR. Chap. Mathematical microworlds and intelligent Computer-Assisted instruction (cf. p. 77, 78).

Travers, Michael David (1996). « Programming with Agents : New metaphors for thinking about computation ». In : (cf. p. 82, 83, 89).

Tricot, André, Fabienne Plégat-Soutjis, Jean-François Camps, Alban Amiel, Gladys Lutz et Agnès Morcillo (2003). « Utilité, utilisabilité, acceptabilité : interpréter les relations entre trois dimensions de l'évaluation des EIAH ». In : *Environnements Informatiques pour l'Apprentissage Humain*, p. 391–402 (cf. p. 21, 22).

Van Staalduinen, Jan-Paul et Sara de Freitas (2011). « A game-based learning framework : Linking game design and learning ». In : *Learning to play : exploring the future of education with video games* 53, p. 29 (cf. p. 12, 27, 29, 31).

Von Wangenheim, Christiane Gresse, Marcello Thiry et Djone Kochanski (2009). « Empirical evaluation of an educational game on software measurement ». In : *Empirical Software Engineering* 14.4, p. 418–452 (cf. p. 9, 25, 191).

Vygotsky, Lev (1967). « Play and its role in the mental development of the child ». In : *Soviet psychology* 5.3, p. 6–18 (cf. p. 11).

Wang, Ting-Chung, Wen-Hui Mei, Shu-Ling Lin, Sheng-Kuang Chiu et Janet Mei-Chuen Lin (2009). « Teaching programming concepts to high school students with alice ». In : *39th IEEE Frontiers in Education Conference*, p. 1–6 (cf. p. 84, 89).

Warren, Scott, Greg Jones et Lin Lin (2011). « Usability and play testing ». In : *Serious educational game assessment*. Springer, p. 131–146 (cf. p. 21).

Willis, Jerry, Larry Hovey et Kathleen Gartelos Hovey (1987). *Computer simulations : A source book to learning in an electronic environment*. Garland (cf. p. 80).

Winnicott, Donald Wood (1971). *Jeu et réalité*. Gallimard (cf. p. 10).

Winograd, Terry (1972). « Understanding natural language ». In : *Cognitive psychology* 3.1, p. 1–191 (cf. p. 77).

Wong, Wee Ling et al. (2007). « Serious video game effectiveness ». In : *Proceedings of the International Conference on Advances in Computer Entertainment Technology*. ACM, p. 49–55 (cf. p. 25).

Woodworth, Pat et Wanda Dann (1999). « Integrating console and Event-Driven models in CS1 ». In : *ACM SIGCSE Bulletin* 31.1, p. 132–135 (cf. p. 35, 47, 50).

Wright, Matthew, Adrian Freed et Ali Momeni (2003). « Opensound control : State of the art 2003 ». In : *Proceedings of the 2003 conference on New interfaces for musical expression*. National University of Singapore, p. 153–160 (cf. p. 95).

Xinogalos, Stelios, Maya Satratzemi et Vassilios Dagdilelis (2006). « An introduction to Object-Oriented Programming with a didactic microworld : objectKarel ». In : *Computers & Education* 47.2, p. 148–171 (cf. p. 4, 35, 49, 54, 69, 70, 72, 82, 84, 89).

Yuan, Wenjie, Chengji Deng, Hongxi Zhu et Jun Li (2013). « The statistical analysis and evaluation of examination results of materials research methods course ». In : *Creative Education* 3.07, p. 162 (cf. p. 203–205).

Zichermann, Gabe et Christopher Cunningham (2011). *Gamification by design : Implementing game mechanics in web and mobile apps*. O'Reilly Media (cf. p. 11).