Apprentissage de la programmation à l'aide d'un jeu sérieux

Mathieu Muratet, Patrice Torguet, Jean Pierre Jessel, Fabienne Viallet

Apprentissage de la programmation à l'aide d'un jeu sérieux

Mathieu Muratet

Patrice Torguet

Jean-Pierre Jessel

IRIT – UMR 5505
Section 27 - Informatique
muratet@irit.fr, torguet@irit.fr, jessel@irit.fr

Fabienne Viallet

DiDiST CREFI-T Didactique des Disciplines des Sciences et Techniques Centre de Recherche en Education Formation et Insertion - EA pluri-établissements 799
Section 70 - Sciences de l'Education
fabienne.viallet@iut-tlse3.fr

Université de Toulouse, Paul Sabatier
118 route de Narbonne, 31062 Toulouse Cedex 9
+33 (0) 5 61 55 66 11

MOTS-CLÉS :

Jeux sérieux, programmation, stratégie temps réel.

RÉSUMÉ :

Ce document présente un prototype de jeu sérieux dont l'objectif est d'inciter le joueur à programmer à l'aide d'un jeu de stratégie temps réel (STR). Dans ce type de jeu, le joueur donne des ordres à ses unités pour réaliser des opérations (se déplacer, construire un bâtiment…). Actuellement, ces instructions sont données à l'aide de la souris en cliquant sur une carte. L'objectif est d'encourager le joueur à donner ces ordres par la programmation. Ce jeu est à destination des étudiants en informatique et peut être utilisé dans le parcours universitaire ou professionnel. Le joueur utilise le C ou le C++ comme langage de programmation

INTRODUCTION

Depuis quelques années, les étudiants délaissent les disciplines scientifiques. Comme le démontrent les travaux de Crenshaw *et al.* (2008) et Kelleher (2006), ce phénomène n'est pas spécifique à la France. Le nombre d'étudiants inscrits en informatique dans notre université suit cette même tendance et a diminué de 16,6% sur les quatre dernières années. Pour tenter d'arrêter et d'inverser cette progression, nous étudions de nouveaux supports pédagogiques susceptibles de dynamiser les apprentissages : les jeux sérieux. Nous pensons que ces

nouvelles technologies peuvent attirer et maintenir les étudiants dans les universités. Ainsi, nous souhaitons utiliser un jeu sérieux pour intéresser les étudiants à l'informatique à travers l'apprentissage de la programmation qui constitue un élément fondamental dans la formation d'un informaticien.

Dans une première partie, nous définissons brièvement le concept de jeu sérieux, puis nous présentons un ensemble de travaux abordant l'apprentissage de la programmation.

Dans une deuxième partie, nous introduisons notre prototype à travers son contenu pédagogique et le choix du support. Enfin, nous décrivons de façon détaillée le fonctionnement de notre application.

1 JEUX SERIEUX

Le terme « jeu sérieux » est très largement utilisé sous de nombreuses dénominations. Ainsi, suivant les définitions, il inclut plusieurs familles d'applications dont voici quelques exemples : l'apprentissage en ligne, le ludo-éducatif, les jeux classiques ou numériques à base d'apprentissage (Susi *et al.* 2007).

Dans tous les cas, le point critique d'un jeu sérieux est la relation entre le jeu et son contenu pédagogique. L'expérience a montré que les jeux sérieux atteignent leurs objectifs s'ils ont une forte composante « jeu » clairement mise en avant. En effet, pour progresser dans un jeu vidéo, le joueur passe par une phase d'apprentissage. Le jeu sérieux exploite cette caractéristique pour instruire le joueur. Cette approche a été utilisée pour *America's Army* (Zyda 2006). Ce jeu sérieux est le premier à avoir remporté un réel succès. Un jeu sérieux est un environnement vidéo-ludique pas uniquement destiné aux enfants mais à un public beaucoup plus large.

Les jeux sérieux sont présents aujourd'hui dans plusieurs secteurs d'activité comme l'éducation, l'administration, la santé, la défense, les entreprises, la sécurité civile et les sciences. Suivant le public considéré, le type de jeu (présentation et contenu) évolue :

- Pour le grand public, les jeux sérieux peuvent être utilisés pour la sensibilisation à des problèmes généraux de santé, de sécurité ou d'environnement.
- Pour l'université ou pour l'entreprise, les jeux sérieux doivent pouvoir fournir un contenu plus complet et précis en fonction du niveau de l'utilisateur. Ils permettent aux apprenants en fin de formation d'aborder et de résoudre des problèmes complexes.
- Pour des formations plus spécifiques comme le pilotage ou la chirurgie, des jeux sérieux à base d'immersion peuvent permettre des simulations physiquement réalistes. Ces jeux s'appuient sur des modèles mathématiques sous-jacents complets, en vue de préparer au mieux les personnes aux situations critiques.

Le jeu sérieux doit donc être conçu en fonction du secteur d'activité, du public et des moyens disponibles (matériels et financiers) pour sa mise en œuvre. Blackman (2005) fait une synthèse sur l'industrie du jeu et ses applications au grand public. Les moteurs graphiques des jeux vidéo, de plus en plus perfectionnés, peuvent être utilisés pour des applications autres que le jeu car ils proposent des rendus temps réels et des « moteurs » physiques réalistes. Des applications d'entraînement, de visualisation interactive et de simulation de situation utilisent largement les technologies des jeux vidéo. Il est donc clair que les jeux aux bases sérieuses et amusantes joueront un rôle important dans un futur proche.

Les jeux sérieux sont en plein essor, mais peu d'entre eux sont conçus pour l'informatique et plus particulièrement pour l'apprentissage de la programmation.

2 TRAVAUX RATTACHES

L'informatique est une vaste discipline possédant de nombreuses spécialités. L'apprentissage de la programmation en est une clé essentielle et incontournable. Pour faciliter cet apprentissage, des logiciels ont été développés.

Certains de ces logiciels utilisent des langages graphiques à base de blocs. Cette métaphore de programmation permet à l'étudiant de se détacher de la syntaxe afin de se concentrer sur l'algorithmique. StarLogo The Next Generation (Klopfer *et al.* 2005), Scratch (Maloney *et al.* 2004), Alice2 (Kelleher *et al.* 2002) et Cleogo (Cockburn & Bryant 1998) s'inscrivent dans cette approche. Ils s'adressent à des personnes n'ayant jamais eu de contact préalable avec la programmation. Ils ont pour objectif d'attirer les étudiants vers l'informatique et de les initier à la logique de la programmation, sans qu'ils aient pour autant une connaissance formelle des concepts.

Une démarche différente consiste à utiliser la compétition pour motiver des étudiants déjà experts en programmation. C'est le cas du projet Robocode[1] et de l'évènement international RoboCup[2]. Tous les deux proposent aux joueurs de programmer des Intelligences Artificielles (IA) en vue de piloter des robots et de les mettre en concurrence au cours de sessions organisées.

La dernière solution utilise le jeu vidéo pour « accrocher » le joueur et l'amener vers la programmation. Le projet WISE (*Wireless Intelligent Simulation Environment*) de Cook *et al.* (2004) est un environnement de jeu interactif qui mélange jeux virtuels et physiques. Colobot[3] est le seul exemple, que nous connaissons, de jeu vidéo complet qui combine interactivité, histoire et programmation. Dans ce jeu, le joueur doit coloniser des planètes en utilisant et programmant des robots. Cependant, ces jeux s'adressent plutôt à des programmeurs confirmés capables d'élaborer des algorithmes sophistiqués inspirés de l'IA.

Si tous ces logiciels sont bien destinés à l'apprentissage et à la pratique éducative de la programmation, quel est leur positionnement par rapport aux jeux sérieux ?

Premièrement, nous ne pouvons considérer StarLogo TNG, Scratch, Alice2, Cloego ou RoboCup comme des jeux sérieux. Pour cela, ces applications ne s'intègrent pas directement dans notre étude, cependant, elles présentent des approches intéressantes et transférables sur un jeu sérieux tel que la programmation à base de blocs ou les environnements collaboratifs.

Deuxièmement, Robocode, WISE et Colobot ne sont pas suffisamment complets. Robocode manque d'interactivité : le joueur est inactif durant la simulation, il reste spectateur de sa propre IA. WISE nécessite de nombreuses ressources (espace, robots…) ce qui rend sa mise en œuvre complexe et ajoute des contraintes à l'expérimentation. Enfin, Colobot manque d'un mode multi joueur. Or, nous pensons que le travail compétitif et collaboratif introduit par l'aspect multi joueur peut être très intéressant pour l'étudiant (Johnson & Johnson 1994).

Ainsi, notre travail s'inspire de ces précédents outils. Nous proposons de soutenir l'apprentissage de la programmation via une plateforme ludique, interactive et multi-utilisateur. Afin de conserver le plaisir du jeu, nous choisissons un jeu vidéo multi joueur existant appartenant à un genre de jeu populaire. Notre contribution consiste à améliorer ce jeu en fournissant la possibilité au joueur de contrôler les entités du jeu grâce à la programmation. En d'autres termes, notre outil est un jeu vidéo multi joueur où la

[1] Robocode : http://robocode.sourceforge.net/

[2] RoboCup : http://www.robocup.org/

[3] Colobot : http://www.ceebot.com/colobot/index-f.php

programmation devient un atout pour le joueur. Nous allons maintenant donner quelques détails sur notre système, pour cela nous présentons le savoir supporté par notre outil et nous analysons les difficultés que rencontrent les étudiants dans l'apprentissage traditionnel de la programmation.

3 ENSEIGNEMENT DE LA PROGRAMMATION

Notre objectif est de fournir un outil de formation complémentaire à l'enseignement classique de la programmation. Ce jeu est à destination des étudiants novices en programmation qui éprouvent des difficultés dans l'apprentissage de la discipline. Selon Janine Rogalski (1988), l'enseignement de la programmation présente en effet des spécificités :

- Les élèves doivent apprendre à passer « du faire » au « faire faire par un ordinateur ».
- Les élèves doivent acquérir des objets spécifiques que sont les structures de contrôles (test, itération et récursivité).
- Les élèves doivent acquérir une représentation des connaissances basée sur des activités de modélisation, notamment mathématiques, qui sont très difficiles à aborder tant par l'enseignant que par l'élève.
- Un programme répond à une exigence de réalisation sur un matériel donné, dans un temps donné, dans un environnement donné.

Ces difficultés constituant souvent des obstacles à l'apprentissage, beaucoup d'étudiants en début de formation se découragent et abandonnent leurs études.

Pourtant, lorsque les concepts spécifiques ont été enseignés, les enseignants considèrent que la pratique de la programmation permet de surmonter les problèmes : « l'acquisition du contenu d'un langage de programmation se présente comme un apprentissage par analogie qui nécessite, pour être mené à bien, des rétroactions apportées par les résultats de la soumission d'un programme à la machine » (Hoc & Mendelsohn, 1987). Ainsi, dans le cadre de la formation traditionnelle, l'enseignement de la programmation est réalisé sous la forme de cours théoriques où sont exposés les concepts, suivis de travaux dirigés où il s'agit d'écrire des algorithmes sur papier et de travaux pratiques effectués sur machines pour tester les algorithmes écrits et analyser les rétroactions. L'approche suivie est celle de la résolution de problèmes.

Pourtant depuis quelques années, les étudiants se plaignent des situations didactiques[4] proposées. En effet, la grande majorité d'entre eux ont une pratique importante des jeux vidéos (86% des étudiants interrogés jouent aux jeux vidéo[5]) et ne comprennent pas pourquoi ils passent, dans le cadre de leur formation, plusieurs semaines à écrire des programmes pour tracer des dessins à l'aide de tirets cadratins à l'écran, alors qu'ils manipulent chez eux, avec les mêmes matériels, des jeux en 3 dimensions.

[4] Les situations didactiques sont « des situations qui servent à enseigner. Elles peuvent êtres considérés soit comme l'environnement de l'élève mis en œuvre et manipulé par l'enseignant qui le considère comme un outil, soit comme l'environnement tout entier de l'élève, l'enseignant et le système éducatif lui-même compris. » (Brousseau, 1998). Nous les considérons ici sous leur première désignation.

[5] Ce sondage a été effectué lors de l'année universitaire 2007/2008 sur un ensemble de 181 étudiants provenant de trois formations différentes (Licence 2ème année Informatique, IUT informatique, IUT Génie Civil).

4 CONTENU PEDAGOGIQUE

L'outil développé permet d'envisager une nouvelle situation didactique, adéquate « dans la mesure où elle permet à l'élève de rencontrer des problèmes qui l'obligent à affronter des difficultés présumées ou connues » (Rogalski, 1987). Il permet aux étudiants novices d'aborder la programmation impérative, mais peut-être également utilisé pour apprendre la programmation orientée objet, événementielle ou parallèle. Les langages manipulés actuellement sont le C ou le C++, langages de références largement utilisés dans l'industrie et enseignés dans les universités.

Du point de vue de l'étudiant, le code saisi et compilé est dynamiquement et interactivement pris en compte par le jeu. Les programmeurs étant novices, il convient donc, de les aider en leur dissimulant toutes difficultés liées à la complexité du moteur du jeu de façon à ce qu'ils puissent se concentrer au maximum sur leurs objectifs : mettre en œuvre les premiers concepts informatiques étudiés. Ainsi la cohérence de la partie est maintenue grâce à un ensemble de mécanismes de liaison et de synchronisation totalement transparent pour lui. De cette manière, l'étudiant peut se concentrer entièrement sur sa programmation et le jeu.

Cet outil pourra également être utilisé à d'autres niveaux d'apprentissage, différents degrés d'abstraction étant disponibles. Par exemple, il est possible de créer pour les débutants une interface minimale permettant de donner de simples ordres. Pour les étudiants plus confirmés, un accès à l'implémentation complète du jeu peut être envisagé afin qu'ils puissent mettre en œuvre des concepts plus complexes et être confrontés à une application sophistiquée.

5 DESCRITPION DU SYSTÈME

Pour supporter notre système, nous utilisons un type de jeu bien connu des joueurs : les jeux de stratégie temps réel (STR). Dans cette catégorie de jeu, le joueur contrôle une armée composée d'unités. Il peut communiquer avec l'environnement virtuel en donnant des ordres à ses unités afin de réaliser des actions (se déplacer, construire un bâtiment…). Actuellement, ces ordres sont donnés en cliquant avec la souris sur une carte, nous souhaitons encourager le joueur à les donner par la programmation.

Nous n'avons pas comme prétention de développer un nouveau moteur de STR. Nous avons donc recherché un jeu existant qui pouvait nous servir de point de départ. Ce jeu doit convenir à nos attentes et doit être capable de supporter nos modifications. Nous n'avons pas pu travailler avec de grands classiques tels que Warcraft III, Age Of Empires III ou bien d'autres jeux car ils sont propriétaires et leurs codes ne sont pas disponibles. Par conséquent, le jeu recherché doit nécessairement être ouvert afin d'avoir accès à son code et de pouvoir l'étudier en vue d'y apporter nos modifications. Heureusement, quelques projets sont en développement avec la volonté d'une diffusion de l'information. C'est le cas d'Open Real-Time Strategy (ORTS) et du projet Spring[6]. Tout les deux sont des jeux de stratégie temps réel multi joueur en 3D (Figure 1).

[6] Spring : http://spring.clan-sy.com/

Figure 1. A gauche : ORTS. A droite : Spring.

ORTS (Buro 2002; Buro & Furtak 2005) est développé pour fournir un environnement de programmation en vue d'étudier des algorithmes liés à l'IA. Ce jeu est donc conçu pour permettre aux programmeurs confirmés de programmer et d'intégrer facilement leur IA au moteur. De plus, ORTS propose un mode multi joueur, cette caractéristique est importante car nous comptons fortement axer notre jeu sérieux sur cet aspect. A plus long terme, nous souhaiterions en effet rendre le jeu sérieux massivement multi joueur et créer un environnement persistant où il serait continuellement possible de programmer de façon ludique.

Récemment, nous avons également intégré notre module dans l'autre moteur de jeu, Spring, pour s'assurer de l'indépendance de notre module vis-à-vis d'ORTS. Ce moteur est en tout point différent d'ORTS tant sur son architecture réseau que sur sa conception interne. Cependant, avec un minimum de modification, nous avons intégré notre module dans ce moteur.

6 ASPECTS TECHNIQUES

Notre objectif est de permettre au joueur de saisir son code et de l'intégrer dans le jeu où il sera exécuté. De cette manière, le joueur pourra suivre le déroulement de son programme à travers le comportement de ses unités dans le jeu vidéo. Mais dans ORTS ou Spring, chaque modification de code implique d'arrêter le jeu et de le recompiler pour que les changements puissent prendre effet à la prochaine partie. C'est le principe des langages de programmation compilés comme le C++ (utilisé dans ORTS et Spring). Dès lors, le premier problème à résoudre est : comment intégrer le code du joueur dans le moteur du jeu sans avoir à l'arrêter et à le recompiler ? Cette amélioration permet une plus grande interactivité car le joueur peut modifier, compiler et intégrer son code sans avoir à arrêter le jeu et ainsi maintenir la progression et la cohérence de la partie.

Pour répondre à ce problème, nous aurions pu choisir un langage de script mais pour des raisons de performances, nous avons choisi d'utiliser une bibliothèque dynamique et de concevoir une interface pour le développement du code.

6.1 Bibliothèque dynamique

La sémantique du terme bibliothèque dynamique résume bien son utilité. La bibliothèque fournit des fonctions qui peuvent être appelées et exécutées par le programme qui la consulte. Quant à la notion de dynamique, elle indique que la bibliothèque pourra être chargée, utilisée et éliminée pendant l'exécution du programme.

Dans notre application, la bibliothèque contient le code saisi par le joueur et définit le comportement de ses unités. Le jeu utilise donc cette bibliothèque pour déterminer les actions à réaliser. A chaque modification de la bibliothèque, la nouvelle IA est rechargée. Grâce à ce principe, le code contenant le comportement des unités est complètement indépendant du jeu. Le joueur peut donc maintenant modifier son code et le recompiler sous forme d'une bibliothèque pour qu'il soit automatiquement intégré au jeu en cours de partie. En règle générale, la bibliothèque se suffit à elle même. Pourtant, dans notre application, elle est censée accéder à la boite à outils du moteur afin de manipuler les données. Il a donc fallu recompiler chaque moteur, ayant servi de tests, pour permettre à la bibliothèque d'accéder aux éléments du jeu.

L'exécution de la bibliothèque est réalisée dans un *thread* (ou processus léger) pour permettre au client de rester actif et apte à réagir aux actions de l'utilisateur ou du serveur. Dorénavant, des IA complexes peuvent être mises en place sans influencer les performances du jeu. En contre partie, la programmation parallèle introduit des difficultés supplémentaires au niveau de la réalisation. En effet, ce type de programmation demande la mise en place de mécanismes entre les différents processus pour permettre d'assurer la synchronisation et la cohérence des données partagées. Cependant le joueur n'a pas conscience de tout ceci.

Nous avons également assuré la fiabilité de notre système en le protégeant contre les bogues générés par les étudiants. En effet, les joueurs étant en apprentissage de la programmation, il est fortement probable que ceux-ci réalisent des erreurs. Ces bogues peuvent causer des erreurs systèmes (erreur de segmentation par exemple) ou des levées d'exceptions. Pour récupérer les erreurs déclenchées dans la bibliothèque nous utilisons les signaux systèmes. Ainsi, lorsqu'une interruption est générée dans le code du joueur, seul le *thread* exécutant le code du joueur est interrompu. Une information est alors donnée au joueur pour l'informer du type d'erreur ayant arrêté son IA. De cette manière, le fonctionnement du jeu n'est pas dépendant des mauvais fonctionnements de l'IA.

L'utilisation de la bibliothèque dynamique possède un avantage supplémentaire. Elle dissimule au joueur la complexité du jeu vidéo. Comme nous l'avons précisé précédemment, le joueur doit saisir le code correspondant au comportement de ses unités. En règle générale, vouloir modifier une partie d'un programme consiste, au préalable, à analyser la structure, l'organisation et le fonctionnement de l'application. La bibliothèque dynamique aide le joueur en extrayant l'IA du jeu. De cette manière, l'utilisateur n'a pas conscience des difficultés liées à l'intégration de son code dans le moteur de jeu.

6.2 Environnement de développement

La bibliothèque dynamique donne au joueur l'opportunité de modifier son code de façon interactive. Elle l'assiste dans son travail en cachant la complexité du moteur. Cependant, l'utilisation du jeu reste fastidieuse en raison des architectures et des arborescences de fichier complexes. Pour aider le joueur encore un peu plus, il est nécessaire de rendre le logiciel plus intuitif. Pour cette raison, nous avons conçu une interface appelée le centre de développement (CDD) (Figure 2).

Le CDD est un composant qui facilite la conception et la manipulation des réalisations du joueur. Il est complémentaire au jeu et simplifie la gestion des projets à travers un ensemble de menus. Cependant, le CDD est indépendant et n'est pas nécessaire au fonctionnement du moteur vice-versa. Il complète la bibliothèque dynamique en simplifiant la manipulation du système de synchronisation et de liaison entre le code du joueur et le jeu. Ainsi, le joueur commence par créer la classique fonction "int main (){...}" comme si son code était indépendant du jeu. Il peut alors utiliser un ensemble de fonctions (définies en relation avec les connaissances du joueur, les objectifs pédagogiques…) pour manipuler les entités du jeu.

Ainsi, le joueur peut facilement compiler et injecter son code dans le moteur et observer les résultats.

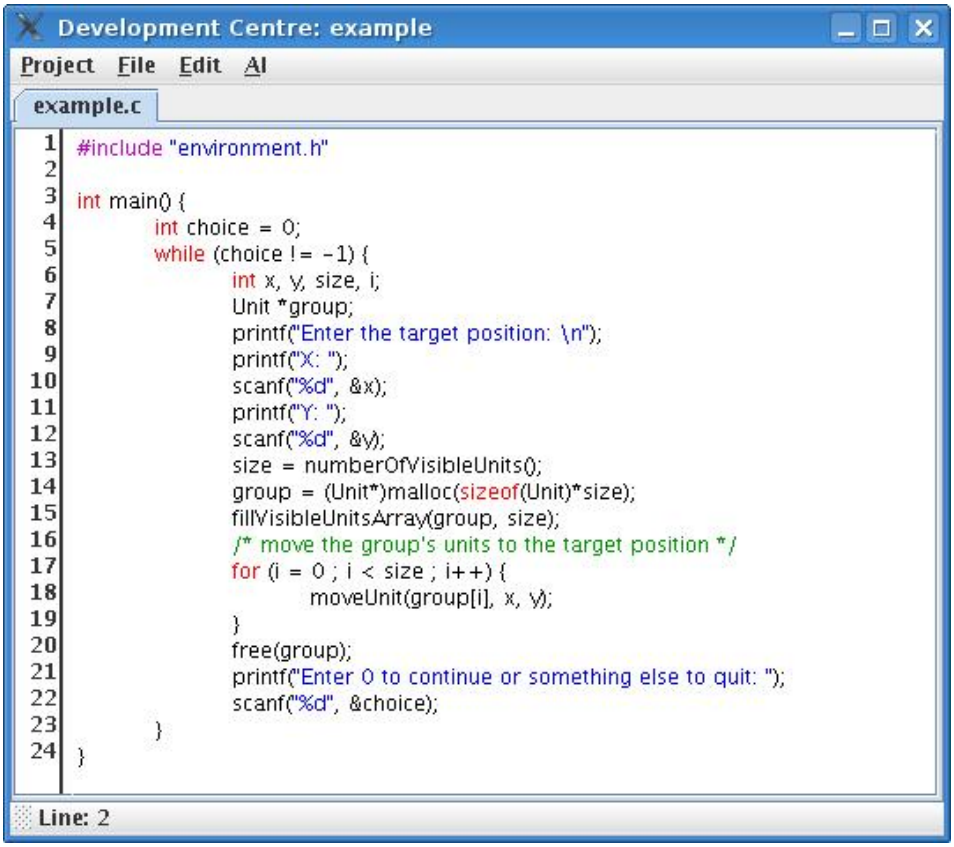

Figure 2. Le centre de développement

6.3 Vue d'ensemble

Notre application est composée de trois entités (Figure 3): le moteur de jeu, la bibliothèque dynamique contenant le code du joueur et le CDD. Il a été nécessaire de modifier le moteur pour permettre au jeu de charger la bibliothèque (à travers le « Chargeur ») et fournir au joueur les outils nécessaires à la manipulation des données (grâce à l'« IMJ »). La bibliothèque dynamique possède une interface (« IGCU ») qui permet son utilisation et un *thread* pour l'exécution du code du joueur. Nous allons maintenant détailler tous ces composants.

« IGCU » signifie « Interface de Gestion du Code de l'Utilisateur ». Elle permet de contrôler l'exécution du code de l'utilisateur (lancer, arrêter).

Le « Chargeur » est conçu pour charger la bibliothèque dynamique lorsque celle-ci est créée ou modifiée. Il a également la responsabilité de la libérer si celle-ci est supprimée. Le « Chargeur » a une autre utilité ; il pilote l'exécution du code de l'utilisateur à travers l'« IGCU » afin de maintenir la même version entre la bibliothèque et le code exécuté.

L'« IMJ » signifie « Interface du Moteur de Jeu » et assure une double fonctionnalité. Premièrement, c'est une interface fournissant à l'utilisateur la possibilité d'interagir avec le jeu, elle sert de point d'entrée au moteur. Deuxièmement, elle assure la synchronisation entre le code du joueur et le moteur du jeu, ceci afin de respecter l'intégrité et la consistance du déroulement de la partie.

Finalement, le CDD permet au joueur de modifier son code contenu dans la bibliothèque et de la (re)construire pour indiquer au moteur que le code a changé et qu'il est temps de le mettre à jour.

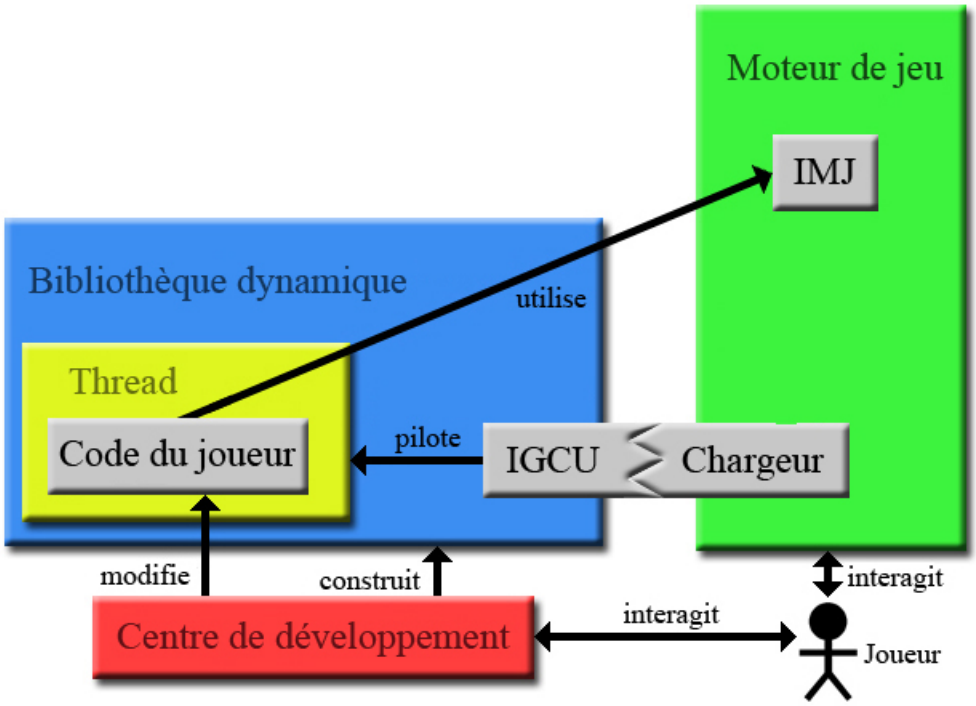

Figure 3. Architecture

7 CONCLUSIONS ET TRAVAUX A VENIR

Dans ce document, nous décrivons un prototype de jeu sérieux dont l'objectif est l'apprentissage de la programmation d'une manière amusante et interactive pour des étudiants novices en programmation. Cette approche est motivée par la baisse du nombre d'étudiant et le fort taux d'abandon en début de cursus informatique. Nous avons donc augmenté plusieurs moteurs de jeu pour qu'ils puissent prendre en compte du nouveau code pendant leur exécution. Ce prototype permet, contrairement à d'autres outils d'apprentissage de la programmation, de manipuler interactivement deux langages de programmation très largement utilisés : le C et le C++. Par ailleurs, nous pensons fournir la possibilité d'utiliser d'autres langages de programmation comme Java ou VBA (*Visual Basic for Applications*) afin d'étendre notre projet à d'autres formations. Ce prototype permet à des étudiants inexpérimentés ou confirmés de s'amuser tout en développant des programmes. De plus, nous avons vérifié que nos modules étaient facilement intégrables dans d'autres moteurs de STR. Ceci permet de changer de jeu en fonction de nos objectifs et de pouvoir suivre l'évolution rapide des jeux vidéo.

La prochaine étape est l'expérimentation. Nous avons commencé à travailler avec des partenaires comme le SUP (Service Universitaire de Pédagogie) de notre université et bien sûr les professeurs et étudiants des formations concernées. Les analyses expérimentales vérifieront son utilisabilité et son efficacité. D'autre part, il sera important de déterminer comment l'utilisateur bascule entre le jeu et le CDD. D'un point de vue didactique, il sera intéressant d'analyser comment l'introduction du jeu vidéo conditionne l'activité de programmation. Une analyse comparative épistémologique des tâches à réaliser dans le

contexte de TP traditionnel et de TP avec l'outil permettra de déterminer ce qui est réellement enseigné.

La majorité des STR fonctionnent sur une architecture P2P (*peer to peer*) où la simulation est dupliquée dans chaque application. A tous les pas de simulation, chaque application synchronise sa simulation avec les autres. Cette architecture n'est pas évolutive et limite le nombre de joueurs. Cependant, ORTS propose une architecture client-serveur. Cette caractéristique est intéressante pour tenter de porter ce moteur vers un système massivement multi joueur. Il serait alors possible à plusieurs centaines de joueurs de partager leurs expériences dans un monde virtuel persistant. De plus, le sujet des MMORTS (*Massively Multiplayer On-line Real Time Strategy*) a très peu été étudié, laissant des perspectives de recherche intéressantes.

Finalement, le CDD facilite l'utilisation du logiciel, mais peut être amélioré pour simplifier un peu plus l'interaction avec le monde virtuel. Il serait alors intéressant de définir un système optionnel de saisie à base de bloc à l'image de Alice2 ou StarLogo The Next Generation. Ceci permettrait d'aider les débutants à se détacher de la syntaxe du C ou du C++.

Toutes ces améliorations permettraient à notre application de devenir un jeu sérieux massivement multi joueur et plus précisément un jeu sérieux de stratégie temps réel massivement multi joueur où la seule limite serait l'imagination du joueur.

Bibliographie

BLACKMAN S.

2005, *Serious games…and less!*, in *SIGGRAPH Computer Graphics*, n°39, vol. I, pages 12-16.

BROUSSEAU G.

1998, *La théorie des situations didactiques*, La pensée sauvage, Grenoble, France.

BURO M.

2002, *ORTS : A Hack-Free RTS Game Environment*, Actes du 3ème colloque international *Computers and Games*, Edmonton, Canada, 25-27 Juillet 2002, Springer : Berlin.

BURO M., FURTAK T.

2005, *On the development of a free RTS game engine*, Actes du 1er colloque annuel *North American Game-On*, Montréal, Canada, 22-23 Août 2005.

COCKBURN A., BRYANT A.

1998, *Cloego: Collaborative and Multi-Metaphor Programming for Kids*, Actes du 3ème colloque *Asian Pacific Computer and Human Interaction*, Shonan Village Center, Japon, 15-17 Juillet 1998, IEEE Computer Society: Washington DC USA.

COOK D. J., HUBER M., YERRABALLI R., HOLDER L. B.

2004, *Enhancing Computer Science Education with a Wireless Intelligent Simulation Environment*, in *Journal of Computer in Higher Education*, n°16, vol. I, pages 106-127.

CRENSHAW T. L., CHAMBERS E. W., METCALF H., THAKLAR U.

2008, *A case study of retention practices at the University of Illinois at Urbana-Champaign*, Actes du 39ème colloque technique ACM *Computer Science Education*, Portland, Oregon USA, 12-15 Mars 2008.

HOC J. M., MENDELSOHN P.

1987, *Les langages informatiques dans l'enseignement*, in *Psychologie Française*, n°32, vol. IV, pages 221-299, Décembre 1987.

JOHNSON R. T., JOHNSON D. W.

1994, *An overview of cooperative learning*, in J. Thousand, A. Villa and A. Nevin (Eds.), *Creativity and collaborative learning*, Baltimore: Brookes Press.

KELLEHER C.

2006, *Alice and The Sims: the story from the Alice side of the fence*, Actes du colloque annuel *Serious Games Summit D.C.*, Washington, DC, USA, 30-31 Octobre 2006.

KELLEHER C., COSGROVE D., CULYBA D., FORLINES C., PRATT J., PAUSCH R.

2002, *Alice2: Programming without Syntax Errors*. Actes du 15ème colloque annuel *User Interface Software & Technology,* Paris, France, 27-30 Octobre 2002.

KLOPFER E., YOON S.

2005, *Developing Games and Simulations for Today and Tomorrow's Tech Savvy Youth*, in *TechTrends: Linking Research & Practice to Improve Learning*, n°49, vol. III, pages 33-41.

MALONEY J., BURD L., KAFAI Y., RUSK N., SILVERMAN B., RESNICK M.

2004, *Scratch: A Sneak Preview*. Actes du 2ème colloque international *Creating Connecting, and Collaborating through Computing*, Keihanna-Plaza, Kyoto, Japon, 29-30 Janvier 2004. IEEE Computer Society: Washington DC USA.

ROGALSKI J.

1988, *Didactique de l'Informatique et acquisition de la programmation*. In *Recherche en didactique des mathématiques*, n°9, vol. III, pages 407-426.

1987, *Acquisition et didactique des structures conditionnelles en programmation informatique.* In *Psychologie Française*, n°32, vol. IV, pages 275-280, Décembre 1987.

SUSI T., JOHANNESSON M., BACKLUND P.

2007, *Serious Games – An Overview.* Rapport technique HS-IKI-TR-07-001, School of Humanities and Informatics, Université de Skövde, Suède, 5 Février 2007.

ZYDA M.

2005, *From Visual Simulation to Virtual Reality to Games*, in *Computer*, n°38, vol. IX, pages 25-32.