Actes de l'atelier APIMU 2023 @ EIAH : Apprendre la Pensée Informatique de la Maternelle à l'Université

Julien Broisin, Christophe Declercq, Cédric Fluckiger, Yannick Parmentier, Yvan Peter, Yann Secq

Atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université », dans le cadre de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH) (APIMU-EIAH) [1]

Actes de l'atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université ».

APIMU 2023

Julien Broisin, Christophe Declercq, Cédric Fluckiger, Yannick Parmentier, Yvan Peter, Yann Secq (Éds.)

Brest, France, 13 juin 2023

1. http://eiah23.apimu.org

Préface

Le Groupe Thématique Apprentissage de la Pensée Informatique de la Maternelle à l'Université (APIMU) s'inscrit dans le cadre de l'Association des Technologies de l'Information pour l'Éducation et la Formation (ATIEF). Il vise à fédérer une communauté de chercheuses et de chercheurs issu · e · s de différents champs disciplinaires autour de la problématique de l'apprentissage de la pensée informatique afin de bâtir une vision commune du domaine, en partager les problématiques et envisager les solutions possibles.

Cet atelier s'est déroulé le 13 juin 2023 à Brest lors de la conférence Environnements Informatiques pour l'Apprentissage Humain (EIAH). Il a été organisé conjointement par le GT APIMU et le projet ANR IE-CARE. Il s'agit de la 6e occurrence d'une série d'ateliers qui se déroule traditionnellement durant la conférence EIAH ou les Rencontres Jeunes Chercheur · es en EIAH (RJC-EIAH). Pour cet atelier nous avons souhaité mettre l'accent sur la mise à l'épreuve des dispositifs et outils proposés : dispositifs expérimentaux, analyses qualitatives et quantitatives des outils et dispositifs mis en œuvre et leurs résultat.

L'atelier s'est déroulé en deux temps avec une présentation d'articles suivi d'une discussion articulée autour de quatre questions présentées dans la suite. Vous trouverez dans ces actes les 5 articles sélectionnés qui sont représentatifs de la diversité des approches et des questionnements :

— Les articles couvrent le primaire/collège (1 article), le lycée et en particulier la spécialité Numérique et Sciences Informatiques (NSI) (3 articles) et l'enseignement supérieur (1 article).
— 4 articles ont une dimension compétences,
— 4 articles évoquent l'analyse de traces,
— 4 articles comprennent des propositions d'outils / plates-formes.

Les questions élaborées par le comité d'organisation pour animer la discussion sont le reflet des thèmes récurrents trouvés dans ces articles :

— En quoi une approche par compétences permet de diversifier les activités proposées aux élèves ?
— Comment valider les compétences ? Automatiquement à partir des traces, ou est-ce une tâche réservée à l'enseignant ?
— Comment favoriser le partage de référentiels de compétences ?
— Quelle accessibilité des traces récoltées par les différentes plateformes ? Comment proposer une base de traces ouverte pour définir des analyses (e.g., pour l'évaluation de compétences) ?

Il ressort des discussions qu'on ne peut pas envisager un modèle de compétences unique car ceux-ci peuvent être influencés par des questions institutionnelles ou des questions de recherche particulières, par exemple. Un travail qui pourrait se développer dans le cadre du GT APIMU serait de recenser et caractériser les référentiels de compétences existants.

Sur la question des traces, il y a un accord sur le fait qu'elles peuvent avoir des usages multiples, pour l'élève, pour l'enseignant · e, pour le chercheur et la chercheuse. La validation automatique des activités n'est pas forcément évidente ni même toujours souhaitable. Fournir des indicateurs aux enseignant · e · s pour qu'ils valident les compétences des élèves est une piste envisageable à condition de disposer d'une modélisation des activités en terme de compétences.

On note que les auteurs ont en général utilisé le format d'échange xAPI pour la génération des traces souvent avec des vocabulaires ad hoc. Un vocabulaire unifié faciliterait le partage des traces et la mise en commun de traitement, ce qui ouvre une autre piste de travail pour le GT APIMU.

En conclusion, la qualité des communications proposées et la richesse des débats qui ont suivi montrent que la communauté qui s'intéresse au sein des EIAH à l'enseignement/apprentissage de l'informatique s'est emparée des sujets liés aux compétences et à l'analyse de traces, ce qui ouvre des perspectives de recherche intéressantes sur l'amélioration des environnements existants voire la création de nouveaux EIAH.

Julien Broisin, Christophe Declercq, Cédric Fluckiger, Yannick Parmentier, Yvan Peter, Yann Secq
(juin 2023)

Comités

Comité d'organisation

— Yvan PETER (Université de Lille)
— Yannick PARMENTIER (Université de Lorraine)
— Yann SECQ (Université de Lille)
— Julien BROISIN (Université de Toulouse)
— Christophe DECLERCQ (INSPÉ de la Réunion)
— Cédric FLUCKIGER (Université de Lille)

Comité scientifique

— Georges-Louis BARON (Université de Paris)
— Florent BECKER (Université Orléans)
— Charles BOISVERT (Université Sheffield)
— Laure BOLKA-TABARY (Université de Lille)
— Isabelle COLLET (Université Genève)
— Kris COOLSAET (Universiteit Gent)
— Yannis DELMAS (Université de Poitiers)
— Brigitte DENIS (Université de Liège)
— Liesbeth DE MOL (Université de Lille)
— Fahima DJELIL (IMT Atlantique)
— Béatrice DROT-DELANGE (Université de Clermont-Ferrand)
— Marie DUFLOT-KREMER (Université de Lorraine)
— Olivier GOLETTI (Université de Louvain)
— Monique GRANDBASTIEN (Université de Lorraine)
— Colin de la HIGUERA (Université de Nantes)
— Sébastien JOLIVET (Université de Genève)
— Bern MARTENS (Universiteit Leuven)
— Kim MENS (Université de Louvain)
— Gabriel PARRIAUX (HEP Lausanne)
— Christophe REFFAY (Université Franche-Comté)
— Robert REUTER (Université du Luxembourg)
— Margarida ROMERO (Université Côte d'Azur)
— Eric SANCHEZ (TECFA, Université de Genève)
— Thierry VIÉVILLE (Inria Sophia Antipolis)
— Amel YESSAD (Sorbonne Université)

Table des matières

Actes de l'atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université »
Brest, France, 13 juin 2023

Apprentissage de la programmation Python - Une première analyse exploratoire de l'usage des tests

Mirabelle Nebut[1] Yvan Peter[1]
(1) Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
mirabelle.nebut@univ-lille.fr, yvan.peter@univ-lille.fr

RÉSUMÉ
Cet article présente un travail qui démarre sur l'usage des tests pour l'apprentissage de la programmation en Python en première année d'université. L'article présente l'instrumentation qui a été réalisée sur l'environnement de programmation `Thonny` ainsi que les premières visualisations issues du traitement de données exploratoire.

ABSTRACT
Here the title in English.

This article presents a starting research work about the use of tests for the learning of Python programming in the first year at university. The article presents the instrumentation developped in the `Thonny` developpement environnement as well as the first visualizations from the preliminary data exploration.

MOTS-CLÉS : Apprentissage de la programmation, tests unitaires, analyse exploratoire des données.

KEYWORDS: Programming learning, unit tests, exploratory data analysis.

1 Introduction

L'apprentissage de la programmation est notoirement difficile et génère beaucoup d'échecs et d'abandons. Il existe donc une recherche conséquente sur le sujet afin de comprendre les difficultés des apprenant·es et leurs comportements de programmation (Luxton-Reilly et al., 2018). Notre travail cible l'apprentissage des tests dans le cadre de l'apprentissage de la programmation en Python, dans le contexte de la première année de licence. Nous souhaitons, à terme, évaluer si l'usage des tests dès l'initiation à la programmation a un effet sur cet apprentissage (positif ou négatif). Nous souhaitons aussi évaluer quel dose de test introduire, entre sensibilisation et apprentissage.

Dans la mesure où les outils de tests existants peuvent être compliqués à mettre en œuvre par des novices en programmation et génèrent une charge cognitive supplémentaire, nous proposons un outil adapté à une initiation : `L1Test`. Celui-ci a été intégré sous forme de greffon à l'environnement de développement `Thonny` (Annamaa, 2015). Nous avons par ailleurs instrumenté `Thonny` afin de capturer des traces d'usage pour nos analyses.

Dans un premier temps nous présentons un état de l'art lié à l'utilisation des tests durant l'apprentissage de la programmation. Nous présentons dans un deuxième temps l'instrumentation réalisée sur `Thonny`. Enfin nous abordons une première exploration des données sur les traces collectées depuis septembre 2022 avant de définir les suites de ce travail.

2 État de l'art : test et apprentissage de la programmation

On trouve dans (Garousi et al., 2020) une synthèse des travaux de recherche existants concernant l'enseignement du test du logiciel dans son sens le plus large et dans (Scatalon et al., 2019) une synthèse exclusivement dédiée aux cours introductifs à la programmation. L'idée générale est que les étudiant·es apprennent traditionnellement le test dans des cours intermédiaires ou avancés, alors qu'il est maintenant largement reconnu qu'il devrait être abordé dès les cours introductifs à la programmation.

Nous nous intéressons pour notre part spécifiquement à l'enseignement du test unitaire intégré à l'enseignement des bases de la programmation avec Python. Dans (Garousi et al., 2020) on trouve une énumération des défis concernant à la fois les enseignant·es et les étudiant·es : les étudiant·es sont souvent peu motivé·es par l'activité de test ; la pratique du test automatisé demande souvent d'utiliser des outils et des méthodes nécessitant de l'expérience en programmation ; l'apprentissage du test augmente la charge cognitive (syntaxe des tests, prise en main de l'environnement de test). Ces deux derniers points sont particulièrement problématiques pour des étudiant·es qui affrontent l'apprentissage des bases de la programmation. Dans (Scatalon et al., 2019) on trouve une liste de bénéfices apportés par l'enseignement précoce du test (ex : le résultat des tests est un retour utile à l'étudiant·e) ainsi qu'une liste d'inconvénients, entre autres la difficulté pour les étudiant·es d'écrire leurs propres tests de manière pertinente et la difficulté à dégager du temps pour l'apprentissage du test (défi aussi présent dans (Garousi et al., 2020)). Les outils et les tâches liées au test doivent rester simples (notamment la syntaxe des tests, l'environnement de test, la méthodologie de développement et les activités liées au test) et s'intégrer à l'apprentissage de la programmation.

Nous nous intéressons ici particulièrement aux outils permettant l'apprentissage du test. Notre cours de première année n'aborde que la programmation impérative, or les bibliothèques de test unitaire utilisent majoritairement l'objet (à l'exception de la bibliothèque `pytest` [1] qui est sans syntaxe objet). Dans des cours plus avancés ou des cours de vérification et validation dédiés au test, l'utilisation d'outils de test réels est pertinent (Garousi et al., 2020). Dans notre contexte un outil dédié à l'apprentissage fait sens. Par ailleurs nous avons choisi d'intégrer notre outil à l'environnement de développement intégré (IDE) `Thonny` utilisé par les étudiants. Nous n'abordons donc pas du tout ici les outils type plate-forme web.

L'apprentissage du test chez des débutant·es en Python se fait maintenant dès le lycée dans la spécialité NSI, avec une problématique d'outillage similaire. Les documents d'accompagnement sur Eduscol pour la classe de première [2] et de terminale [3] mentionnent les outils `pytest` et `doctest` [4] ainsi que l'utilisation du mot-clé Python `assert`. La bibliothèque `pytest` demande d'écrire les tests dans un fichier séparé de celui du programme, à l'intérieur de fonctions dont le nom évoque l'objectif du test. Ce principe peut poser problème aux débutant·es malhabiles dans leur usage d'un gestionnaire de fichier et apprenant tout juste à écrire des fonctions. L'utilisation du mot-clé `assert` à l'intérieur des programmes ne nécessite pas d'apprendre une bibliothèque, mais éloigne beaucoup de l'usage classique des outils de test. En licence nous avons utilisé jusqu'à la rentrée 2022 l'outil `doctest` (inclus dans la distribution Python standard) : cet outil permet d'écrire des tests basiques très simplement dans la documentation des fonctions Python (docstring), en réutilisant la syntaxe

1. https ://pytest.org/
2. https ://eduscol.education.fr/document/30058/download
3. https ://eduscol.education.fr/document/7298/download
4. https ://docs.python.org/3/library/doctest.html

de l'interpréteur Python (voir aussi la section 3.1). Ce principe a été repris pour Java dans l'outil `comTest` (Lappalainen et al., 2010).

3 Instrumentation

Nous utilisons depluis plusieurs années l'IDE `Thonny` (Annamaa, 2015) dédié à l'apprentissage de la programmation avec Python. `Thonny` propose des fonctionnalités d'apprentissage utiles et un débogueur simple d'accès. L'exécution des programmes et les interactions avec l'interpréteur Python se font à l'intérieur de `Thonny`. Nous souhaitions intégrer l'activité de test à celle de programmation et donc l'inclure de même à l'intérieur de `Thonny`.

Les fonctionnalités de `Thonny` peuvent être étendues par un mécanisme de greffons que nous avons utilisé. La figure 1 présente l'interface globale de l'environnement ainsi que les deux greffons ajoutés qui sont décrits dans la suite. `Thonny` se présente sous la forme d'une interface graphique (Tkinter) avec une fenêtre principale contenant une fenêtre d'édition de code, une fenêtre contenant une console Python, ainsi que des boutons permettant notamment d'exécuter le contenu de l'éditeur.

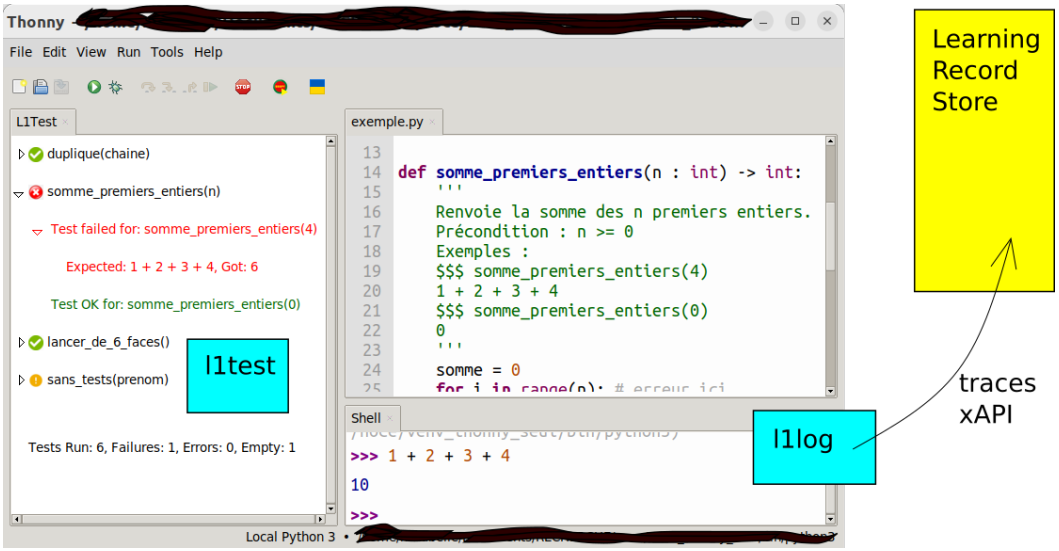


FIGURE 1 – Environnement `Thonny` et ses greffons

3.1 Greffon de test pour `Thonny` L1test

L'outil `L1Test` s'inspire de `doctest` quant au principe d'écrire les tests unitaires directement dans la docstring des fonctions. Ces exemples servent à la fois de documentation pour le code et de tests exécutables. Comme `doctest`, la syntaxe de `L1Test` reprend très simplement la syntaxe de l'interpréteur Python, l'invite `>>>` ayant été changée en `$$$` pour ne pas mélanger les 2 outils. La figure 2 montre des tests pour des fonctions `somme_n_premiers_entiers` et `affiche_prenom` avec `doctest` et `L1Test`.

Le greffon `L1Test` ajoute à la fenêtre principale de `Thonny` un bouton permettant de lancer les tests (icône à gauche du drapeau ukrainien, figure 1) et à gauche une fenêtre dans laquelle s'affichent les verdicts des tests. Les tests qui passent apparaissent en vert, les tests qui ne passent pas apparaissent

```
>>> liste_somme_premiers_entiers(3)
[1, 3, 6]
>>> affiche_presentation("Paul")
Je suis Paul
```

```
$$$ liste_somme_premiers_entiers(3)
[1, 1 + 2, 1 + 2 + 3]
$$$ affiche_presentation("Paul")
None
```

FIGURE 2 – Exemples de tests avec `doctest` et `L1Test`

en rouge (conventions des bibliothèques de test classiques). Les fonctions sans test apparaissent en orange. Par contraste `doctest` est uniquement textuel dans la console, en noir et blanc.

Le différence avec `doctest` est sémantique. Le principe de `doctest`, initialement non dédié au test unitaire, est compliqué à comprendre à un stade d'initiation. Un test `doctest` passe uniquement si la valeur attendue est, comparée caractère par caractère, celle que fournirait l'interpréteur Python. Dans le test de la figure 2, l'évaluation de `liste_somme_premiers_entiers(3)` produit la liste `[1, 3, 6]`. Le test échouera si la valeur attendue est écrite `[1,3,6]` faute de caractère espace. Il est donc conseillé d'écrire puis exécuter le code de la fonction puis de copier la sortie dans la docstring. Cela incite à écrire les tests après le code, ce qui va à l'encontre des approches préconisant de commencer par les tests. De plus `doctest` permet de "tester" une procédure qui affiche une chaîne de caractère sur la sortie standard (cf `affiche_presentation` en figure 2), ce qui n'aide pas à lever la traditionnelle confusion des débutant·es entre `print` et `return`.

L'outil `L1Test` utilise au contraire l'opérateur de comparaison d'égalité == de Python pour comparer les valeurs attendue et obtenue. Les listes `[1,2]` et `[1, 2]` sont ainsi équivalentes. Il est possible d'utiliser une expression pour la valeur attendue (cf figure 2). Enfin une procédure d'affichage, qui renvoie la valeur `None` comme toute fonction Python sans instruction `return`, ne pourra être testée que via cette valeur et est non testable.

3.2 Greffon de collecte de traces L1log

Les étudiant·es génèrent un gros volume de données pendant les activités de programmation, quand ils ou elles interagissent avec l'environnement de programmation. Un grand nombre d'articles s'intéressent à la collecte et à l'analyse de ces données pour en tirer des conclusions sur le comportement des étudiant·es (Luxton-Reilly et al., 2018). Nous souhaitions récolter des informations sur l'utilisation de `L1Test` (nombre de clics sur le bouton "Run test", nombre de tests présents dans les programmes, etc) mais aussi sur la manière dont le test s'insère dans les pratiques de programmation des étudiant·es. Nous avons collecté des traces d'usage dépassant la seule utilisation de `L1Test`.

`Thonny` génère un événement Tkinter pour chaque interaction de l'utilisateur avec l'interface. Ces événements peuvent être enregistrés sous forme de fichiers JSON en local et un greffon permet de rejouer la session de programmation. Toutefois, ce niveau de trace est trop fin pour nos objectifs (chaque caractère ajouté dans l'éditeur génère un événement) et les traces doivent être exportées manuellement. Pour cette raison nous avons ajouté à `Thonny` un greffon (`L1Log`) chargé de filtrer, d'abstraire des événements plus sémantiques (par ex. exécution du code) et de les envoyer en temps réel dans un dépôt de données (*Learning Record Store* – LRS) au format xAPI [5] (Figure 3).

xAPI est un format de donnée JSON standardisé pour la représentation des activités d'apprentissage. Un enregistrement xAPI est composé a minima d'un *acteur* (l'apprenant), réalisant une action (le

5. `https://xapi.com/`

verbe, par ex. soumettre une réponse) sur un *objet* (par ex. un quiz). Les verbes permettant de décrire l'action ne sont pas définis par xAPI et sont tirés de vocabulaires extérieurs.

ProgSnap2 (pour *Programming Snapshot*) est une spécification pour la représentation et le stockage des données issues d'activités de programmation (Price et al., 2020). Cette spécification offre un vocabulaire spécifique aux activités liées à la programmation (manipulation de fichiers, compilation, exécution d'un programme, etc.).

Nous avons donc d'une part un format standard liés aux activités d'apprentissage, pour lequel il existe déjà des outils de stockage et de traitements et d'autre part une spécification *ad hoc* qui correspond aux activités qui nous intéressent. Nous avons combiné les deux en intégrant le vocabulaire de ProgSnap2 pour décrire les verbes et objets de nos traces dans le format xAPI. La figure 3 montre un exemple de trace pour un test (verbe `Run.Test` tiré de ProgSnap2). Le tableau 1 recense les propriétés collectées pour chaque événement.


```
{
 "timestamp":{"$date":"2022-10-07T11:15:44.981Z"},
 "actor":
  {
   "openid":"https://www.cristal.univ-lille.fr/users/281b59ea7d35e20",
   "objectType":"Agent"
  },
 "verb":{"id":"https://www.cristal.univ-lille.fr/verbs/Run.test"},
 "object":
 {
  "id":"https://www.cristal.univ-lille.fr/objects/Test",
  "objectType":"Activity",
  "extension":{"https://www.cristal.univ-lille.fr/objects/File/Filename":"file_-8712108631165266730",
  "https://www.cristal.univ-lille.fr/objects/Test/TestedExpression":"maximum(4, 1)",
  "https://www.cristal.univ-lille.fr/objects/Test/TestedLine":27,
  "https://www.cristal.univ-lille.fr/objects/Test/TestedFunction":"maximum"}
 },
 "result":
 {
  "success":true,
  "extension":{"https://www.cristal.univ-lille.fr/objects/Test/expectedResult":"4",
  "https://www.cristal.univ-lille.fr/objects/Test/obtainedResult":"4"}
 }
}
```


FIGURE 3 – Exemple de trace xAPI pour un test.

Propriété	**Obligatoire**	**Signification**
identifiant utilisateur	oui	hash basé sur le login de l'étudiant
identifiant session	oui	session `Thonny`, de son lancement à son arrêt
timestamp	oui	horodotage de l'événement
type d'événement	oui	correspond aux actions effectuées dans l'éditeur
résultat	oui	vrai si l'action s'est déroulée sans erreur, faux sinon
buffer	non	contenu de l'éditeur au moment de l'action
sortie	non	sortie produite lors de l'exécution d'un programme, des tests ou de l'utilisation du shell

TABLE 1 – Propriétés collectées pour les événements

4 Collecte et analyse exploratoire

4.1 Contexte de la collecte de données

L'expérimentation s'est déroulée lors du premier semestre de l'année universitaire 2022-2023 au sein de la première année du portail SESI. Ce portail SESI regroupe des étudiant·es se destinant à diverses licences scientifiques (Informatique, Chimie, etc). L'origine des étudiant·es et leur niveau en programmation est très hétérogène : bacs français avec ou sans spécialité NSI, bacs étrangers, réorientations. La collecte a eu lieu uniquement dans les groupes de TP de 2 amphis, soit environ 300 étudiant·es. Les autres groupes de TP ont utilisé la version de `Thonny` non instrumentée (sans collecte des traces) et `doctest`. Il est donc impossible de comparer l'utilisation des outils `doctest` et `L1Test`. Il est probable que certain·es enseignant·es, absent·es à la réunion de présentation de `L1Test`, ont fait utiliser `doctest`. Par ailleurs la collecte a souffert de divers défauts de conception de `L1Log` et `L1Test`, qui ont tout de même permis le déroulement des TPs.

4.2 Analyse des traces

Nous avons pour le moment travaillé sur le nettoyage des traces suite aux différents défauts de `L1Log` et `L1Test` et autres aléas de collecte (certains étudiant·es ont par exemple utilisé par erreur la version instrumentée de `Thonny` alors que leur groupe utilisait `doctest`). Les premières analyses exploratoires que nous avons menées ont donné des résultats encore très partiels. 162 étudiants ont utilisé `L1Test` au sens où au moins une de leur session contient des actions de lancement des tests avec le bouton dédié (verbe `Run.test`). La figure 4 montre l'usage du lancement des tests au cours des 11 semaines du semestre [6]. On constate une utilisation non négligeable de `L1Test` durant tout le semestre. La décroissance observée en seconde partie de semestre suit la décroissance générale des actions dans `Thonny`, corrélée au décrochage progressif d'une partie de la promotion.

La figure 5 présente une visualisation qui permet d'étudier le comportement des étudiant·e·s durant une session. Chaque trait représente une action de l'étudiant·e : celles dont le résultat a provoqué une erreur sont au dessous de l'axe (incluant les tests en échec) et les autres sont au dessus. Sur l'exemple présenté, l'étudiant·e lance beaucoup les tests durant la session avec de nombreux tests en échec en début de session.

5 Perspectives

Cette première phase d'expérimentation nous a permis de tester les greffons développés sur un grand volume d'étudiant·es et de corriger des erreurs sur `L1Test` et `L1Log`. Elle nous a également permis d'avoir une première étude des données produites ainsi que des visualisations. L'outil `L1test` a comme ambition de permettre l'écriture de tests unitaires basiques de manière aussi simple qu'avec `doctest`, avec une sémantique et une interface graphique plus intuitives. Nous souhaitons évaluer les effets de la sensibilisation au test avec `L1Test` sur l'apprentissage de la programmation (notamment la correction du code) et définir des indicateurs en ce sens, dans le cadre de notre expérimentation.

6. `L1Test` n'a pas été utilisé pendant la semaine 37, première semaine des enseignements. Les semaines 44 et 45 correspondent à une vacance des enseignements.

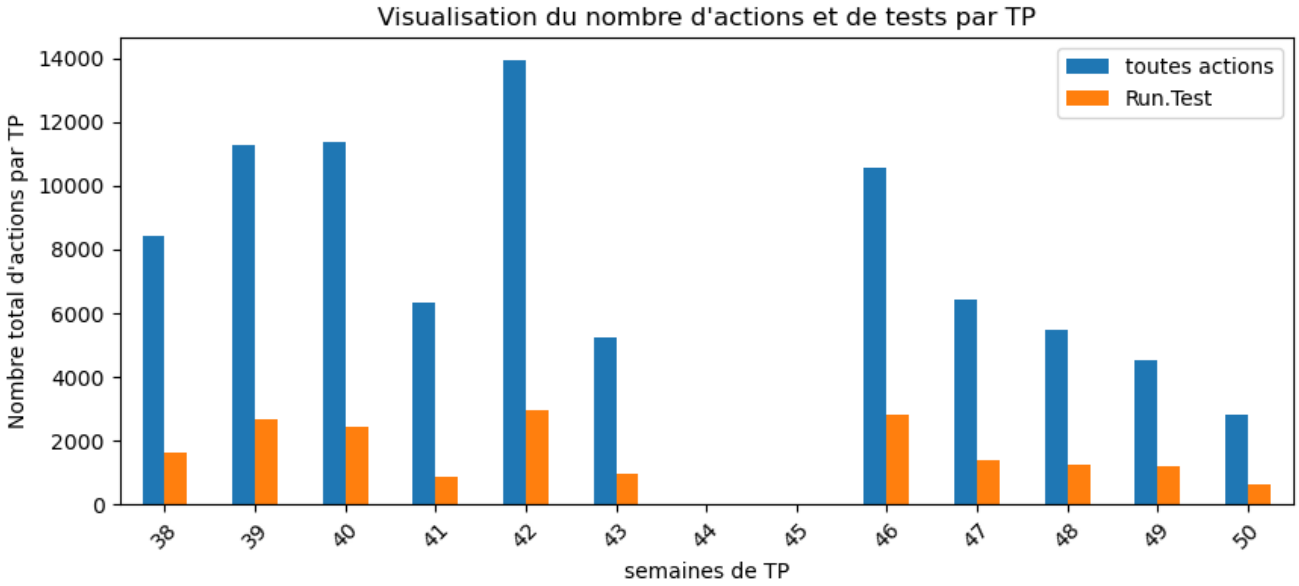


FIGURE 4 – visualisation globale de l'usage des tests.

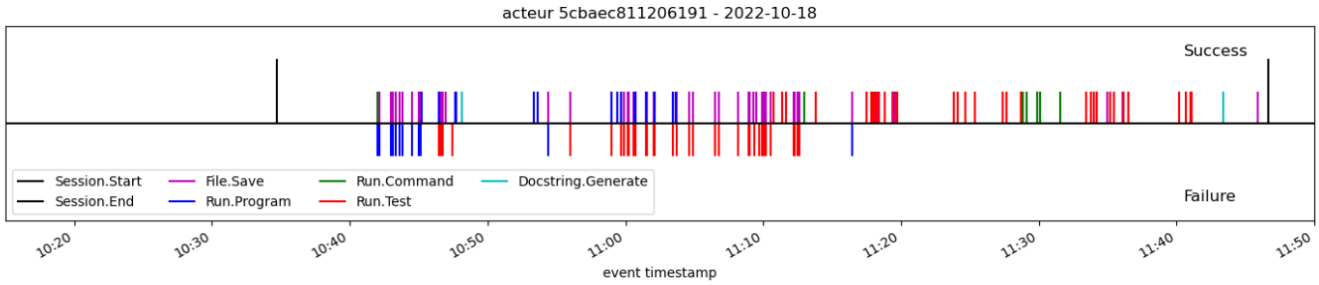


FIGURE 5 – Exemple de frises temporelles.

Nous aimerions comprendre si et à quel moment l'introduction du test répond à un besoin des étudiant·es ou leur apporte une plus-value importante à leur yeux. Parmi les critères à doser, on pourra citer la méthodologie (nous avons choisi de faire écrire les tests avant le code) et le fait de demander aux étudiant·es d'écrire leur propres tests (vs les fournir tout ou partie).

Écrire les tests avant le code permet de porter son attention sur l'utilisation qui sera faite de la fonction et de se représenter son comportement. En particulier le fonctionnement de `L1Test` peut permettre d'aborder par les tests la traditionnelle confusion entre retour et effet de bord. De plus une grosse fonction est difficilement testable : l'utilisation des tests peut permettre d'aborder des questions de conception liées au découpage en fonctionnalités d'un programme. La non-testabilité des procédures d'affichage devrait aussi inciter les étudiant·es à séparer celles-ci des fonctionnalités de calcul.

Exécuter les tests régulièrement fournit un retour formalisé et automatisé sur la présence d'erreurs dans le code. Ce retour semble ressenti comme motivant et autonomisant par les étudiant·es, surtout si la suite de tests est fournie et constitue un indicateur raisonnable de correction du code. Les tests permettent d'ailleurs de rendre plus concrète cette notion de correction, charge aux enseignante·s d'indiquer clairement que les tests prouvent la présence d'erreurs dans le code mais jamais leur

absence, ni le respect des conventions de codage.

Par ailleurs, le Test Driven Learning (Janzen and Saiedian, 2006) est une approche pédagogique entièrement basée sur les tests, beaucoup plus poussée que la nôtre, qui encourage les étudiant·es à écrire leurs propres tests notamment pour passer d'une technique d'essai-erreurs à une réflexion sur le comportement du code (Edwards, 2004). Nous aimerions évaluer si l'écriture des tests engage effectivement les étudiant·es dans une telle démarche de réflexion, et si fournir une suite de tests incite au contraire à faire juste "passer les tests" de manière erratique. Enfin, nous aimerions évaluer via un suivi de cohorte si la sensibilisation au test aura un impact sur l'apprentissage du test unitaire objet classique avec JUnit en 2ème année.

Sur le plan de l'analyse des traces, nous aimerions maintenant commencer à mettre en place un tableau de bord rassemblant différents indicateurs et visualisations afin de faciliter l'analyse du comportement des étudiant·es aussi bien sur le long terme que dans le cadre des TP, dans l'optique de fournir aux étudiants des rétroactions adaptées.

Références

Annamaa, A. (2015). Thonny, : A python ide for learning programming. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, page 343. Association for Computing Machinery.

Edwards, S. H. (2004). Using software testing to move students from trial-and-error to reflection-in-action. In *Proceedings of the 35th SIGCSE technical symposium on Computer science education*, pages 26–30.

Garousi, V., Rainer, A., Lauvås Jr, P., and Arcuri, A. (2020). Software-testing education : A systematic literature mapping. *Journal of Systems and Software*, 165 :110570.

Janzen, D. S. and Saiedian, H. (2006). Test-driven learning : intrinsic integration of testing into the cs/se curriculum. *Acm Sigcse Bulletin*, 38(1) :254–258.

Lappalainen, V., Itkonen, J., Isomöttönen, V., and Kollanus, S. (2010). Comtest : a tool to impart tdd and unit testing to introductory level programming. In *Proceedings of the fifteenth annual conference on Innovation and technology in computer science education*, pages 63–67.

Luxton-Reilly, A., Simon, Albluwi, I., Becker, B. A., Giannakos, M., Kumar, A. N., Ott, L., Paterson, J., Scott, M. J., Sheard, J., and Szabo, C. (2018). Introductory programming : A systematic literature review. In *Proceedings Companion of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018 Companion, page 55–106. Association for Computing Machinery.

Price, T. W., Hovemeyer, D., Rivers, K., Gao, G., Bart, A. C., Kazerouni, A. M., Becker, B. A., Petersen, A., Gusukuma, L., Edwards, S. H., and Babcock, D. (2020). Progsnap2 : A flexible format for programming process data. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*, pages 356-–362. ACM.

Scatalon, L. P., Carver, J. C., Garcia, R. E., and Barbosa, E. F. (2019). Software testing in introductory programming courses : A systematic mapping study. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 421–427.

Actes de l'atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université »
Brest, France, 13 juin 2023

Description et analyse de trois EIAH d'apprentissage de Python

Sébastien Jolivet[1, 2] Eva Dechaux[1] Anne-Claire Gobard[3] Patrick Wang[4]
(1) IUFE, Pavillon Mail, 40 bd du Pont-d'Arve, CH-1205 Genève, Suisse
(2) TECFA, villa Battelle, 7 route de Drize, CH-1227 Carouge, Suisse
(3) Lycée Kastler, 26 avenue de la Palette, 95011 Cergy Pontoise, France
(4) Haute école pédagogique du canton de Vaud, Avenue de Cour 33, CH-1014 Lausanne, Suisse
sebastien.jolivet@unige.ch, eva.dechaux@unige.ch,
anne-clair.gobard@ac-versailles.fr, patrick.wang@hepl.ch

RÉSUMÉ

Dans cet article nous proposons une analyse de trois EIAH d'apprentissage de Python : AlgoPython, Citizen Code Python et Pyrates. Après avoir présenté le fonctionnement de chaque EIAH nous revenons sur les savoirs travaillés dans ces environnements et l'organisation de ce travail. Puis nous étudions les aides et rétroactions présentes ainsi que les possibilités d'orchestration proposées. Ce travail permet de mettre en évidence les points communs et différences entre environnements et soulève diverses questions sur les apports en ce qui concerne les apprentissages. Nous proposons en conclusion une synthèse de ces analyses et diverses exploitations possibles de ce travail.

ABSTRACT

Description and analysis of three Python learning environments

In this article we propose an analysis of three Python learning environments : AlgoPython, Citizen Code Python and Pyrates. First, we present the functioning of each EIAH, the knowledge worked in these environments and the organization of this work. Then we study the hints and feedbacks proposed, and the orchestration possibilities. This work allows us to highlight the similarities and differences between these environments and raises various questions about their contributions in terms of learning. In conclusion, we synthesize these analyses and several possible uses of this work.

MOTS-CLÉS : Environnements d'apprentissage de Python ; analyse d'EIAH ; EIAH.

KEYWORDS: Python learning environments ; learning environment analysis ; TEL.

1 Introduction

Dans cet article nous analysons trois EIAH d'apprentissage de Python. Nous présentons les caractéristiques principales de ces environnements, puis nous détaillons l'étude des savoirs travaillés. Cette étude a été débutée dans (Jolivet et al., 2023). Elle est basée sur un référentiel de types de tâches relatif au travail global de la programmation et au travail spécifique sur les variables, boucles et instructions conditionnelles. Puis nous enrichissons cette première analyse par l'étude des choix didactiques réalisés dans ces environnements, des aides et rétroactions proposées aux apprenants, et des outils d'orchestration mis à disposition des enseignants. La conclusion nous permet de mettre en perspective ces différents éléments pour esquisser des propositions relatives à la conception d'EIAH.

2 Présentation des trois EIAH

Les EIAH étudiés, AlgoPython [1], Citizen Code Python [2] et Pyrates [3], visent aux premiers apprentissages de Python (ils sont nommés AP, CCP et Py dans la suite du texte). Ils proposent tous une version gratuite complète. AP propose un système de licence, à destination des établissements scolaires, offrant des possibilités d'administration de la plateforme et de suivi du travail des élèves. Ils sont tous accessibles en ligne et aucun ne propose d'installation locale. Dans les sections suivantes, nous présentons le fonctionnement de chacun de ces environnements.

2.1 AlgoPython

AP est développé par un professeur de mathématiques et distribué par Génération 5. Dans sa présentation, il est signalé qu'il couvre l'intégralité du programme de lycée en matière d'apprentissage de Python, pour la part intégrée aux mathématiques. Les contenus sont organisés autour de deux grands domaines : l'apprentissage de Python et l'utilisation de Python dans le cadre d'exercices de mathématiques.Celui relatif à l'apprentissage de Python est structuré en huit séries : les instructions, les boucles `for`, les variables, les fonctions, les conditions, les boucles `while`, les listes. Chaque série contient un ensemble de 7 à 12 exercices, complétés par 3 à 12 exercices "bonus". Pour chaque exercice il y a trois zones affichées en permanence : 1) la première contient un énoncé textuel qui indique la tâche à réaliser et contient parfois certains éléments de "cours" ; 2) zone de saisie du code Python ; 3) la console Python qui présente le résultat de l'exécution du code et d'éventuels messages d'erreur. L'accès aux exercices "obligatoires" d'une section est soumis à la réussite du précédent. Enfin, AP propose une console permettant de l'utiliser comme un IDE.

2.2 Pyrates

Cet EIAH présente la particularité d'avoir été développé comme outil d'étude dans le cadre d'un travail de thèse [4] (Branthôme, 2021), il est donc normal qu'il propose ni le même volume de ressources ni la même couverture du domaine que les deux précédents. Cependant, son utilisation auprès d'un nombre important d'élèves et le travail didactique qui a fondé sa conception, motivent son intégration dans les EIAH analysés. Il se positionne dans le moment particulier de la transition entre le collège et le lycée français avec le passage de Scratch (ou autre environnement de programmation par blocs) à Python. L'environnement est organisé sous forme d'un jeu à huit niveaux. À chaque niveau le pirate se déplace dans un environnement, avec lequel il peut interagir, pour atteindre et ouvrir un coffre.

Dans Pyrates, quatre zones sont affichées en permanence : 1) une zone textuelle qui contient *l'objectif* qui définit la tâche à réaliser, des *contraintes* à respecter dans le niveau, et la définition des nouvelles *fonctions de contrôle* utiles pour le niveau ; 2) un tableau qui définit la situation initiale du niveau (le pirate, le coffre et les différents éléments du type clé, les obstacles). L'exécution du programme

1. `https://www.algopython.fr`
2. `https://www.futureengineer.fr/`
3. `https://py-rates.fr`
4. Nous analysons dans cette contribution la version disponible en ligne. D'autres fonctionnalités ont été développées par le concepteur de l'environnement à des fins de recherche et seront peut-être disponibles ultérieurement, mais ce n'est pas le cas à la date de cette analyse.

permet de mettre en action les éléments de cette zone ; 3) une zone de saisie du code Python ; 4) une zone nommée *Mémo Programmation Python* qui est un portail d'accès vers différentes aides.

2.3 Citizen Code Python

Cet EIAH a été développé par la société Tralalère et IOI France avec le soutien d'Amazon Future Engineer. Il propose une « initiation à la programmation pour tous ». Cet environnement présente la spécificité de proposer, pour tous les exercices, de les réaliser en Python ou en Blockly. Il est structuré en 11 *saisons*, chaque saison étant constituée d'un ensemble de 3 à 10 exercices. Tous les exercices sont basés sur la même mécanique : un "robot pince" saisit des blocs situés en dessous et peut les déplacer vers la droite ou la gauche pour reconstituer un modèle ; parfois il est possible de déposer les blocs dans un broyeur qui détruit des éléments.

Trois zones sont affichées en permanence : 1) une zone qui contient l'énoncé textuel qui permet de définir l'objectif à atteindre en spécifiant la position attendue des briques ; 2) une zone avec le "robot pince" et les briques à déplacer, elle participe aussi à la définition de la tâche à réaliser puisque les objets à déplacer y sont dans la configuration initiale ; 3) une zone de saisie du code Python.

Suite à cette présentation globale, nous abordons les savoirs travaillés dans les trois environnements et les choix didactiques pour organiser ce travail.

3 Savoirs travaillés dans ces EIAH et modalités de travail

Du point de vue des thématiques travaillées, AP est le plus complet avec l'ensemble de thèmes suivant : les instructions, les boucles `for`, les variables, les fonctions, les conditions, les boucles `while`, les listes. CPP n'aborde pas les listes tandis que Py n'aborde ni les listes ni les fonctions.

Au-delà de l'identification des thèmes abordés, nous nous intéressons maintenant à la manière dont ils sont travaillés. Pour cela nous reprenons le travail présenté dans (Jolivet et al., 2023) qui identifie un ensemble de types de tâches relatifs à la programmation [5] d'une manière globale et aux différents thèmes communs aux trois EIAH, à savoir les boucles, les variables et les instructions conditionnelles.

3.1 Les types de tâches présents dans les EIAH

Nous ne reprenons pas ici en détail le processus d'identification et de structuration des types de tâches. Nous précisons simplement qu'ils ont été identifiés dans divers moyens d'enseignement de l'informatique du niveau collège et lycée (français et suisse) et qu'il s'agit donc de types de tâches "réels" et non pas "inventés". Nous reprenons simplement, dans les tableaux 1 et 2, la répartition de ces types de tâches dans les trois EIAH. Ces tableaux ont été obtenus en procédant à la résolution systématique de tous les exercices concernés par les trois notions étudiées, puis à l'analyse des solutions. Nous nous sommes limités aux types de tâches prescrits et n'avons pas intégré ceux qui peuvent ou doivent être mobilisés pour la réalisation de la tâche. Par exemple, pour produire une

5. Non pas la programmation dans toute sa généralité, mais dans les limites de ce qui est travaillé lors des premiers apprentissages en milieu scolaire.

TABLE 1 – Effectif des types de tâches de *Programmation* par EIAH

Types de tâches		AP	CCP	Py
	Analyser un code existant	0	0	0
Feuille blanche	Copier et exécuter	0	6	0
	Concevoir un programme	26	43	8
Code existant : Modifier un programme P en P' qui réalise une tâche t	P est erroné	1	0	0
	P réalise t' proche de t	15	0	0
	P contient des trous	3	0	0
Nombre d'exercices décrits		45	49	8

boucle `while` il va notamment être nécessaire *d'affecter une valeur à une variable, initialisation*, cependant nous ne comptabilisons pas ce deuxième type de tâches si la tâche porte sur le premier.

Nous explicitons certains de ces types de tâches en même temps que nous commentons ces tableaux. Les détails sont disponibles dans (Jolivet et al., 2023).

Ce qui est tout d'abord notable dans ce tableau 1, c'est la très faible variété des types de tâches proposés. Ainsi CCP et Py proposent exclusivement [6] le type de tâches *Concevoir un programme* dans une situation de type *feuille blanche*. C'est-à-dire que l'élève ne dispose que de l'énoncé et d'une zone de saisie vide pour réaliser la tâche demandée. AP propose un peu plus de variété puisqu'un peu plus de la moitié des exercices proposés sont aussi de ce type mais qu'un tiers sont du type *Modifier un programme qui réalise une tâche proche ou une partie de la tâche souhaitée*, exercices pour lesquels il est proposé dans la zone énoncée un code presque identique à celui à produire. Enfin, dans AP il y a aussi, mais de manière anecdotique, un exercice où il faut corriger un code donné erroné et trois autres où il faut compléter un code donné qui contient des trous.

Il n'y a par contre, tous environnements confondus, aucun exercice visant l'analyse d'un code existant.

TABLE 2 – Effectif des types de tâches relatifs aux *boucles, variables et instructions conditionnelles*

Type de tâches	AP	CCP	Py
Produire une boucle bornée	21	105	10
Utiliser la valeur d'une expression	5	0	0
Affecter une valeur à une variable, initialisation	9	24	3
Affecter une valeur à une variable, modification	0	13	0
Utiliser une variable dans une expression	6	0	3
Ajouter une ou des instructions à une boucle existante	2	0	0
Concevoir une IC	12	16	2
Déterminer les instructions à exécuter à chaque itération	1	2	0
Modifier les instructions qui font évoluer la condition d'arrêt	1	0	0
Produire une boucle non bornée	6	12	2

Le tableau 2 nous permet de dresser divers constats. Tout d'abord, ce sont essentiellement les types de tâches "principaux" qui sont mobilisés : *Produire une boucle bornée*, *Concevoir une IC*, *Produire une boucle non bornée*. Les éléments qui sont nécessaires à la réalisation de tels types de tâches, par exemple *Déterminer les instructions à exécuter à chaque itération* ou *Utiliser la valeur d'une*

6. CCP propose pour chaque premier exercice d'une série un type de tâches un peu particulier qui est de recopier le code solution du problème qui est donné *in extenso.*

expression, sont évidemment travaillés lors de ces tâches mais ne font que très peu l'objet d'exercices spécifiques. On peut donc estimer que ces environnements permettent de s'entraîner à manipuler les notions fondamentales mais pas réellement d'apprendre le travail avec ces notions puisque les bases permettant l'apprentissage ne font pas l'objet d'exercices spécifiques.

On peut aussi noter la sur-représentation du type de tâches *Produire une boucle bornée* dans CCP. Ceci est la conséquence du choix des auteurs de l'environnement de ne pas autoriser de paramètre pour les fonctions *gauche()* et *droite()* qui permettent de déplacer la pince. Chaque déplacement de plus de deux mouvements doit donc être réalisé à l'aide d'une boucle *for* pour respecter le nombre limite de lignes de code autorisé.

À propos des fonctions il est notable que, avant même d'en faire un objet de travail, elles sont largement utilisées. En particulier dans CCP et Py qui sont basés sur des déplacements, et donc utilisant massivement des fonctions pour gérer ces déplacements. Ainsi dans CCP (*resp.* Py) nous pouvons comptabiliser, sur les exercices étudiés, 399 (*resp.* 27) appels de fonctions sans paramètre et 27 (*resp.* 6) appels de fonctions en passant une valeur en paramètre.

Si cette utilisation des fonctions est rendue nécessaire par le contexte de l'environnement, il est notable que cette utilisation précoce des fonctions n'est nullement exploitée lorsque les fonctions deviennent l'objet du travail dans CCP.

3.2 Choix et artifices didactiques

Dans les trois environnements une fois une série traitée, il est possible que la notion abordée soit remobilisée dans les séries suivantes. Il y a donc *de facto* un choix d'organisation des apprentissages porté par l'environnement. Concernant les choix réalisés dans Pyrates, ils sont largement documentés par son concepteur et nous renvoyons donc à (Branthôme, 2021) pour la présentation de ceux-ci.

Dans le cas de AP et CCP on peut noter comme similitude que les boucles `while` sont toutes les deux traitées à la suite des autres notions, seules les listes pour AP et les booléens pour CCP sont traités après. Dans AP nous pouvons noter la chronologie "boucle `for` ; variables ; fonctions ; conditions" alors que dans CCP la chronologie proposée est "boucle `for` ; instruction conditionnelle ; variables ; fonctions". Dans aucun de ces deux environnements les auteurs ne motivent ces choix, et il n'y a pas à notre connaissance, de travaux de didactique de l'informatique qui permettraient de valider ou invalider l'un plutôt que l'autre.

Au sein des exercices, l'enjeu principal des artifices didactiques utilisés, est de contraindre l'utilisation d'une notion plutôt qu'une autre (par exemple le `for` plutôt que la répétition d'une même séquence d'instructions ou une boucle `while` plutôt qu'une boucle `for`). AP et CCP utilisent à la fois la limitation du nombre de lignes de code et parfois le blocage de l'utilisation d'un mot (par exemple le `for`). Si Py utilise aussi la limitation du nombre autorisé de lignes de code on peut noter l'utilisation intéressante d'un artifice supplémentaire. Dans le cadre de l'utilisation du `while`, une variable aléatoire modifiée à chaque exécution du code est introduite dans la définition de la mission à remplir par le pirate. En l'occurrence la solidité d'un tonneau sur lequel il faut taper pour le détruire afin d'avancer, il devient ainsi nécessaire de *taper tant que le tonneau est présent*.

3.3 Conclusion sur les savoirs travaillés

Nous pouvons constater après analyse des exercices de ces trois EIAH, que ces derniers demandent à l'apprenant, principalement, de concevoir un programme, majoritairement à partir d'une feuille blanche. Il n'y a par ailleurs aucun exercice qui relève de l'analyse d'un code existant. Ces choix peuvent apparaître comme surprenant, car cette dernière activité n'est pas particulièrement difficile à implémenter par exemple sous forme d'exercices d'appariements entre code et problème. Par ailleurs, l'analyse d'un code existant est nécessaire, que ce soit pour adapter un code existant à un nouvel objectif ou pour corriger un code déjà produit.

Par ailleurs, les notions sont le plus souvent travaillées de manière globale sans que les différents types de tâches nécessaires pour leur mise en œuvre ne soient travaillés de manière spécifique. Par exemple, dans le travail sur les boucles, il n'y a aucun exercice qui permet de "simplement" déterminer le nombre d'itérations ou l'expression booléenne à définir.

D'autre part, hormis dans AP, il n'y a pas d'exercices à trous qui permettent pourtant d'amener l'apprenant à travailler spécifiquement une notion, ou une partie de notion, sans être bloqué par le fait de gérer tout ce qui va autour (déclaration des variables, etc.). Il est notable que dans AP, pour les séries sur les fonctions et les listes, il y a une part égale d'exercices de type "feuille blanche" et "d'exercices à compléter avec le début du code donné". Toujours dans AP, l'auteur change très significativement de stratégie dans les séries d'exercices relatifs aux mathématiques, puisque, s'il n'y a toujours pas d'analyse de code, ce sont exclusivement des exercices pour lesquels le début du code est donné (souvent le nommage et le choix des paramètres de la fonction à produire) ou tout le code est donné avec des trous à compléter.

CCP et Py, qui sont des environnements de jeux graphiques, mobilisent de manière significative le type de tâche "appeler une fonction". AP semble mobiliser des tâches plus variées, ce qui peut être expliqué par la présence à la fois d'exercices graphiques avec un robot et d'exercices faisant écrire des mots dans la console.

4 Aides, rétroactions et outils d'orchestration

En complément de l'analyse des savoirs travaillés réalisée dans la section précédente, nous présentons maintenant les apports respectifs des différents environnements relativement aux *rétroactions et aides* et aux possibilités *d'orchestration et de suivi du travail des élèves* disponibles pour un enseignant dans le cadre d'un usage scolaire de l'environnement.

4.1 Aides et rétroactions

Dans cette section, nous nous intéressons aux moyens mis à disposition de l'apprenant pour l'aider dans la résolution de sa tâche. Ceux-ci sont de trois natures : moyens techniques proposés par l'environnement du type exécution pas à pas ; ressources de type "cours" et enfin rétroactions.

Lors de l'exécution du code, tous les EIAH proposent une modalité qui permet d'agir sur la vitesse d'exécution du code. CCP propose trois modes d'exécution : un mode *lecture* durant lequel l'instruction exécutée est en surbrillance dans le code ; un mode *pas à pas* qui permet de visualiser l'instruction en cours d'exécution et son effet sur la scène et un mode qui permet d'afficher directement le résultat

suite à l'exécution du code. Py dispose d'un mode identique au mode *lecture* avec une vitesse d'exécution réglable. AP ne propose qu'un curseur permettant de régler la vitesse d'exécution du code pour quelques exercices (dessin et déplacement d'un robot).

Concernant les aides à disposition de l'apprenant, dans AP elles sont situées dans les énoncés des exercices qui servent aussi à définir les notions et à présenter des exemples associés. Tout est donc à disposition de l'apprenant en permanence, mais rien n'est proposé pour attirer son attention sur tel ou tel élément particulier. Dans Py la zone *Mémo programmation Python* permet d'accéder à différents éléments de cours qui contiennent à la fois des définitions et des exemples. Il y a la particularité, conformément au positionnement de Py, de mettre en parallèle des morceaux de code en Scratch et le code équivalent en Python. Enfin, dans CCP, deux boutons donnent accès l'un à une ressource appelée *Documentation* qui contient des informations générales sur ce qu'est programmer et des informations spécifiques en lien avec le thème travaillé. L'autre bouton donne accès à des indices qui sont liés à la tâche à réaliser : deux à trois indices, progressifs, sont accessibles à la demande de l'apprenant.

Concernant les rétroactions on notera qu'elles sont essentiellement relatives aux aspects syntaxiques. Dans CCP certaines erreurs (oubli de parenthèses lors de l'appel d'une fonction par exemple) sont signalées dès l'écriture du code alors qu'il faut attendre l'exécution dans AP ou Py. Dans Py ou CCP il y aussi des rétroactions contextuelles à l'environnement (déplacement ou action impossible). D'une manière générale on peut noter l'absence de rétroactions épistémiques et le renvoi aux éléments d'aide présentés précédemment en fonction de l'activité de l'apprenant.

4.2 Dispositifs d'orchestration et de suivi pour l'enseignant

CCP est visiblement conçu pour un usage personnel avec la délivrance d'*open badges* en fonction du travail réalisé. Il n'y a aucune fonctionnalité pensée pour l'utilisation dans le cadre scolaire. Le suivi de la progression d'un élève, de ses erreurs, etc., serait donc totalement à la charge de l'enseignant sans dispositif dédié. Un dispositif d'inscription et d'identification permet de conserver, individuellement, la trace du travail réalisé.

Py ne propose pas non plus de dispositif d'inscription de classe ou de suivi global du travail des élèves. Il n'est pas non plus possible de s'inscrire, mais un code est attribué à chaque partie commencée, ce qui permet de reprendre le travail à partir du dernier niveau validé.

Dans AP, sous réserve de souscrire une licence, un enseignant peut créer des classes et ajouter des élèves en leur créant des comptes. Les possibilités d'orchestration sont limitées au déblocage global d'un ou plusieurs thèmes qui peut être décidé au niveau de la classe entière ou élève par élève. Il n'y a pas de gestion au niveau des exercices. Enfin, une interface de suivi de l'activité des élèves est disponible. Même si son ergonomie peut être améliorée, elle permet d'avoir une vision globale de la classe avec le nombre d'exercices terminés par thème pour chaque élève. Un accès au suivi individuel de chaque élève permet d'obtenir quelques précisions sur l'activité de celui-ci.

5 Choix, questions, manques et perspectives

En conclusion nous proposons deux grandes orientations pour prolonger ce travail d'analyse.

Sur l'apprentissage de la programmation

On constate tout d'abord que ces environnements proposent des choix spécifiques, notamment sur la chronologie d'étude des notions. Ils ne semblent pas relever de contraintes internes aux environnements. On peut donc supposer qu'ils sont l'expression des conceptions des auteurs de ces environnements sur l'apprentissage de la programmation. L'impact de ces choix didactiques est un questionnement fondamental à aborder. Pour cela, on pourra s'appuyer sur ces EIAH, par exemple en comparant les apprentissages entre des utilisateurs de CCP et de AP comme un moyen d'étudier l'impact de l'ordre de rencontre avec les différentes notions sur les apprenants.

Un deuxième questionnement, relatif à l'apprentissage de la programmation, porte sur la ou les exploitations possibles de ces environnements en situation de classe. Au delà de la présence de fonctionnalités d'orchestration et de suivi des élèves, ces environnements semblent plutôt destinés à des apprenants qui disposent déjà de certaines bases, a minima d'algorithmique, et une familiarité minimale avec ce que signifie programmer. Ceci soulève la question des moments de l'apprentissage (Chevallard, 1999) pour lesquels ils peuvent s'avérer efficients, rejoignant ici le questionnement posé dans (Jolivet and Grugeon-Allys, 2022). Ils sont sans doute plus adaptés à un travail d'entraînement qu'à une exploitation pour une première rencontre avec les concepts ou l'institutionnalisation par exemple. Ceci est encore renforcé par l'absence de rétroactions épistémiques.

Sur la conception d'EIAH d'apprentissage de la programmation

Dans (Feitelson, 2023) il est signalé la part très significative de l'analyse de codes dans l'activité des développeurs, ceci amène au moins à s'interroger sur la place que devrait avoir une telle activité dans l'apprentissage de la programmation. Or on a constaté le faible nombre de types de tâches différents proposés, et en particulier l'absence d'exercices de lecture / analyse de code. Une diversification des types de tâches proposés, par exemple des activités de lecture de code ou des exercices de type *Parsons Problems* (Ericson et al., 2018), est sans doute une orientation pour développer des EIAH d'apprentissage de la programmation.

Références

Branthôme, M. (2021). Apprentissage de la programmation informatique à la transition collège-lycée. *Revue STICEF*, 28(3). Medium : pdf Publisher : STICEF Version Number : 1.

Chevallard, Y. (1999). L'analyse des pratiques enseignantes en théorie anthropologique du didactique. *Recherches en Didactique des Mathématiques*, 19(2) :221–265.

Ericson, B. J., Foley, J. D., and Rick, J. (2018). Evaluating the efficiency and effectiveness of adaptive parsons problems. In *Proceedings of the 2018 ACM Conference on International Computing Education Research*, pages 60–68.

Feitelson, D. G. (2023). From Code Complexity Metrics to Program Comprehension. *Communications of the ACM*, 66(5) :52–61.

Jolivet, S., Dechaux, E., Gobard, A.-C., and Wang, P. (2023). Construction et exploitation d'un référentiel de types de tâches d'apprentissage de la programmation. In *Actes du colloque EIAH2023 : 11ème Conférence sur les Environnements Informatiques pour l'Apprentissage Humain*, Brest.

Jolivet, S. and Grugeon-Allys, B. (2022). Modélisation de parcours d'apprentissage adaptés à l'apprenant dans un EIAH. In Florensa, I. and Ruiz Munzón, N., editors, *Pré-actes de la 7e conférence internationale sur la théorie anthropologique du didactique (CITAD7)*, pages 92–106, Barcelone.

Actes de l'atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université »
Brest, France, 13 juin 2023

Enseigner SQL en NSI

Emmanuel Desmontils Laura Monceaux
LS2N, 2 rue de la Houssinière, 44322 Nantes, France
emmanuel.desmontils@univ-nantes.fr, laura.monceaux@univ-nantes.fr

RÉSUMÉ

Dans cet article, nous présentons une démarche systématique d'exploration des situations didactiques qui peuvent être envisagées pour enseigner SQL en lycée. En effet, partant du constat que les propositions, aussi bien dans la littérature, les manuels scolaires ou les ressources en ligne, se contentent de proposer des exercices uniquement de la forme "écrire une requête qui recherche...", nous cherchons à diversifier les types de questions. A la suite de ces réflexions, nous développons une application web permettant à un enseignant de proposer des exercices sur diverses bases de données.

ABSTRACT

Teaching SQL

In this article, we present a systematic approach to exploring the didactic situations that can be considered to teach SQL in high school. Indeed, starting from the observation that proposals, both in literature, textbooks or online resources, offer exercises only in the form "write a query that find...", we proposes to diversify the types of questions. Following these reflections, we are developing a web application allowing a teacher to offer exercises on various databases.

MOTS-CLÉS : Didactique en informatique, Bases de données, SQL, NSI.

KEYWORDS: Computer Science Didactic, Databases, SQL.

1 Introduction

L'enseignement des bases de données est présent dans les curricula universitaires depuis très longtemps et, depuis 2019, constitue un chapitre à part entière du programme de NSI [1] de terminale ((MENJ, 2019), annexe page 5). On constate que l'enseignement s'axe sur la compréhension et l'usage des bases de données en général ainsi que sur le questionnement et la modification des données d'une base de données relationnelle. La modélisation d'une base de donnée n'est pas au programme de cette formation, contrairement à toutes les initiations aux bases de données de l'enseignement universitaire. C'est regrettable, car la modélisation permet de mieux comprendre la structure du modèle relationnel, d'améliorer l'efficacité des requêtes SQL produite et de rendre plus autonomes les élèves dans le cadre de la mise en place des projets (part importante des activités de NSI).

Une difficulté particulière en bases de données est le langage SQL utilisé pour exploiter et manipuler les données. C'est un nouveau langage pour les élèves de NSI, mais, en plus, c'est un nouveau paradigme. En effet, le langage de requête SQL n'est pas très complexe en soit, surtout sur le programme de terminale, mais ce changement de paradigme peut troubler voire poser des problèmes

1. "Numérique et Sciences Informatiques" : spécialité qui peut être choisie par des lycéens (cycle secondaire général) en France pour une durée hebdomadaire de 4h en première et de 6h en terminale

de compréhension fondamentaux (Sadiq et al., 2004; Ahadi et al., 2015). Nous proposons donc un ensemble de onze situations didactiques visant à faciliter la découverte et l'apprentissage de ce langage.

Dans la suite de cet article, nous aborderons d'abord l'enseignement de SQL (section 2). Puis, nous présenterons une catégorisation des situations didactiques pour enseigner SQL (section 3). Enfin, nous présenterons l'outillage associé que nous développons (section 4) avant de conclure.

2 Enseignement de SQL

SQL (Structured Query Language) est un langage apparu en 1974 permettant de décrire, renseigner et interroger une base de données relationnelle. SQL comporte quatre parties : le langage de définition de données (LDD), le langage de manipulation de données (LMD), le langage de contrôle de données (LCD) et le langage de contrôle des transactions (LCT). Comme mentionné dans (MENJ, 2019), seule la partie LMD est au programme NSI, allant des requêtes de mise à jour à l'interrogation d'une base de données.

Les travaux autour de l'enseignement de SQL se situent sur deux axes :

- l'étude des erreurs selon des critères qualitatifs (Sadiq et al., 2004; Reisner, 1981; Smelcer, 1995; Miedema et al., 2021) et quantitatifs (Ahadi et al., 2015; Reisner, 1981) mettant en évidence des difficultés linguistiques, syntaxiques (SQL) et sémantiques.
- l'organisation d'entrainements ou d'évaluations avec des réflexions sur l'organisation des activités (entrainements ou évaluation) ou la manière dont saisir les requêtes SQL.

C'est sur ce deuxième axe que notre attention s'est portée. Les travaux abordent peu les types de question que l'on peut présenter pour ces activités. (Reisner, 1981) propose six tâches utilisées pour "mesurer la facilité d'utilisation d'un langage de requête" : "query writing", "query reading", "query interpretation", "question comprehension", "memorization" (être capable de mémoriser et reproduire une base de données) et "problem solving" (étant donné un problème et une base de données, proposer les intentions pour résoudre en problème). Cependant, l'auteur ne développe pas leurs caractéristiques et propriétés, son travail étant axé sur la comparaison des différents langages de requête de l'époque (SQL, QBE, SQUARE & TABLET) pour des développeurs professionnels.

Nous proposons dans cet article, une catégorisation en onze situations didactiques pour permettre l'enseignement de SQL pour des élèves de NSI (voire des étudiants de premier cycle universitaire) où les quatre premières tâches de (Reisner, 1981) sont présentes. Il est nécessaire de considérer ces situations didactiques pour enseigner SQL au regard de la complexité des opérateurs. Plusieurs travaux ont montré la difficulté de certains opérateurs comme le distinct, la jointure (en particulier l'auto-jointure), les sous requêtes (en particulier synchronisées) ou les regroupements (hors NSI) (Kearns et al., 1997; Kleerekoper and Schofield, 2018; Mitrovic, 1998; Renaud and Van Biljon, 2004; Ahadi et al., 2015; Miedema et al., 2021). Il faudra donc considérer que la complexité des situations didactiques que nous allons présenter sera pondérée par les opérations relationnelles qui seront mises en œuvre. En combinant la complexité des situations avec la complexité des opérateurs, il est alors possible de proposer un ordre des questions d'un exercice.

3 Situations didactiques en Base de Données

L'enseignement des bases de données n'est pas récent. Cependant, force est de constater que les exercices proposés ont souvent la même forme : "Donnez une requête qui ...". Dans cette section, nous allons proposer une classification des questions possibles sur SQL. Pour illustrer celle-ci, nous utiliserons la base de données de (Chrisment et al., 2008) (p.62) qui permet de recenser les fleuves et les espaces maritimes dont le modèle relationnel est présenté ci-dessous (les clé primaires sont soulignées et la clés étrangères précédées de #) :

- Pays(id_p,nom_p, superficie, nbhab) ;
- EspaceMaritime(id_em, nom_e, type) ;
- Fleuve(id_f, nom_f, longueur, #id_p, #id_em) ;
- Cotoie(#id_p, #id_em) ;
- Parcourt(#id_p,#id_f, distance).

Trois éléments entrent en jeu lors dans la construction de questions sur SQL :

- l'intention (I) visée par la requête : texte en langage naturel décrivant ce que l'on cherche dans la base de données ;
- la table (T) résultat de la requête : ensemble des tuples ;
- la requête SQL qui peut prendre deux formes : complète (R) ou partielle (r).

L'*intention* donne le besoin fonctionnel de la recherche (ce que l'on veut) et non une traduction en langage naturel de la requête. Une intention est donc un élément clé lors de la construction et la manipulation de la base de données. Elle participe à la construction du schéma (on doit pouvoir faire...) et, bien évidemment, elle est implémentée sous la forme d'une (voire plusieurs) requêtes SQL permettant d'extraire des informations de la base vers l'application. A noter que (Reisner, 1981) nomme cela *question*, mais, dans notre contexte, il y a ambiguïté sur ce terme avec les questions d'un exercice qui englobe l'intention, les attendus, etc. Par exemple, une intention liée à la base de données exemple peut être la phrase : "Le nom des fleuves de longueur supérieure à 1000 Km". La requête partielle répondant à cette intention pourrait être [2] :
"Select ? ? ? ? ? From Fleuve Where longueur > 1000 ;"

A partir des trois éléments, nous avons considéré les différentes combinaisons possibles en faisant varier ce qui est proposé à l'élève et ce qui est attendu de l'élève. Nous avons mis en évidence onze situations didactiques potentiellement intéressantes dont font partie quatre situations de (Reisner, 1981). Nous avons choisi de coder chaque situation sous la forme "Entrées"→"Sortie", où chaque information peut être un élément de l'interrogation d'une base de données : I, T, R ou r. Les entrées sont les informations données à l'élève et la sortie est l'information qu'il devra produire. Quand une entrée est entre crochet, celle ci est optionnelle. Par exemple, "[I]R→T" décrit une situation où l'intention (éventuellement) et la requête sont données à l'élève et celui ci doit donner la table résultat. Pour certaines situations, il est possible d'ajouter une étape "⇒T" qui permet d'obtenir le résultat en exécutant la requête sur le SGBD-R, permettant à l'élève de valider sa production en sortie.

Dans la suite de cette section, nous supposerons que l'élève a connaissance du modèle relationnel et, en cas d'attente de la table résultat, un exemple d'instance de la base de données. Nous évaluerons les différents cas au regard de la complexité de la situation.

2. "? ? ? ?" est la partie à compléter pour répondre à la question.

3.1 Différentes situations didactiques proposées

Transformer un besoin fonctionnel en une requête SQL : I[T]→R

Cette situation didactique est, de loin, la plus courante dans la littérature, les cours en lignes, etc (voir section 3.2). Sans la table, c'est la situation "query writing" de (Reisner, 1981). L'enseignant propose une intention I à l'élève, accompagnée ou non de la table résultat T, et ce dernier doit trouver une requête R qui permet d'y répondre. Il doit être capable de transformer un besoin fonctionnel en une requête SQL.

Par exemple, on peut donner à l'élève l'intention suivante « les fleuves de longueur supérieure à 1000 Km » et il devra fournir la requête suivante en sortie :
"Select nom_f From Fleuve Where longueur > 1000 ;"

Attention, il faut, dans cette situation, insister sur la vérification de la requête. Ce n'est pas parce qu'une requête donne la table résultat attendue T qu'elle est bonne ! De même, ce n'est pas parce qu'une requête ne donne pas de résultat qu'elle est fausse. Donc, il faut proposer des situations qui suivent le déroulement IT1→R⇒T2 avec T2=T1 quand c'est possible.

Mais d'autres situations didactiques pourraient être mises en place pour l'élève avant d'être capable de transformer le besoin fonctionnel en une requête SQL.

Traduire le résultat d'une intention : I→T

Dans cette version, plus simple que la précédente, on demande la table résultat T d'une intention I. C'est la situation "question comprehension" de (Reisner, 1981). Elle est un peu utilisée chez (Balabonski et al., 2020) (voir section 3.2). L'élève n'a pas à connaitre SQL, mais doit proposer les tuples que la requête SQL répondant à l'intention devrait retourner. Cette situation permet d'amener l'élève à traduire un besoin en un ensemble d'actions (parfois informelles). Ces actions servent ensuite à l'enseignant pour introduire le langage SQL et ses différents opérateurs.

Pour faciliter le travail de l'élève, on pourrait décomposer les situations présentées. Par exemple, I→T peut être un travail préalable à la forme I[T]→R, ce qui donnerait I→T→R. L'enseignant demande d'abord la table réponse T à une intention I avant de demander la requête elle-même R. L'élève peut ainsi prendre le temps de comprendre la base de données avant de construire la requête. Cette solution peut être intéressante dans le cas de requête ne donnant aucun résultat. Les élèves peuvent avoir un blocage sur ce type de requête pourtant correcte. Cependant, pour les convaincre, il faut sans doute revenir sur le schéma précédent à savoir I→T1→R⇒T2 (en espérant que T1=T2). Cette démarche peut être généralisée à la vérification de toute situation de ce type.

Comprendre le fonctionnement des opérateurs utilisés par une requête : [I]R→T

La situation suivante est de mettre à disposition de l'élève une requête SQL bien formée R. L'élève doit, à partir d'une instance de la base de données qui lui est proposée, donner la table résultat T de cette requête. Sans l'intention, c'est la situation "query interpretation" de (Reisner, 1981). Cette situation est un peu utilisée chez (Balabonski et al., 2020) (voir section 3.2). L'objectif est de l'amener à comprendre le fonctionnement des opérateurs utilisés par la requête. Le motif de cette situation est "Soit la base de données suivante, donner le résultat de la requête... [qui recherche...]". Par exemple, "Donner la table résultat de la requête suivante qui recherche les fleuves de longueur supérieure à 1000 Km" pour la requête "Select..." (forme IR→T)

Cette situation est aussi intéressante pour amener l'élève à comprendre l'architecture de la base de

données relationnelle. Il faut proposer des requêtes simples, mais aussi des jointures clé-primaire/clé-étrangère pour l'amener à "parcourir" les tables. Dans cette situation, comme dans la précédente, l'enseignant peut amener l'élève à vérifier sa réponse en suivant la forme : IR→T1, R⇒T2 en espérant T1=T2.

Dans cette famille de situations, il est aussi possible de proposer une requête et plusieurs tables possibles. L'élève doit choisir celle qui correspond au résultat de la requête. Cette situation est plus facile que la précédente et peut être proposée en amont.

Découvrir les différents opérateurs disponibles dans une requête SQL : Ir→R & Tr→R

Cette situation est assez classique en programmation, beaucoup moins en bases de données. Elle est totalement absente des ouvrages consultés et est présente rarement en ligne (voir section 3.2). L'idée est de proposer une requête à trous. L'élève doit proposer des solutions. La requête seule est une situation assez difficile, et pas nécessairement intéressante. Il faut donc l'accompagner soit de l'intention I, soit de la table résultat T. Cette approche permet de découvrir les différents opérateurs disponibles dans une requête SQL.

La situation suivante propose de découvrir simplement la projection en proposant pour l'intention "Les fleuves qui se jettent dans une mer" la requête suivante à compléter : "Select ? ? ? ? ? From Fleuve Where longueur > 1000 ;"

Selon le niveau des élèves, il est possible de compliquer un peu cette situation en proposant aussi les opérateurs inutilisés. La place libre doit alors le rester. Par exemple, sur l'exemple précédent, la situation devient : "Select ? ? ? ? ? From Fleuve Where longueur > 1000 Order By ? ? ? ? ? ;"

On peut aussi fournir la table résultat T et la requête précédente (sans l'intention), mais cela est assez délicat. Il est assez complexe de trouver la bonne requête, juste en ayant à sa disposition le résultat, car, bien évidemment, plusieurs requêtes peuvent donner le même résultat.

Comprendre le besoin fonctionnel d'une requête : R[T]→I

Sans la table, c'est la situation "query reading" de (Reisner, 1981). Nous avons ici une situation un peu particulière. Une requête R est proposée (éventuellement avec la table résultat T) et les élèves doivent trouver l'intention I sous-jacente. L'objectif est d'amener les élèves à comprendre le besoin fonctionnel d'une requête pour éventuellement la corriger. La présence de la table (ou la possibilité donnée à l'élève d'exécuter la requête) permet d'aider à trouver cette intention. Cette table peut-être fournie dans un second temps en cas de difficulté à trouver la bonne intention (aide progressive).

En pratique, cette situation n'est pas évidente à mettre en place, car beaucoup d'élèves vont formuler la requête en langage naturel et non énoncer son intention. Par exemple, pour la requête "Select nom_f From Fleuve Natural Join Espace-Maritime Where type = 'mer' ", les élèves diront *"on fait la jointure entre Fleuve et Espace-maritime, on ne garde que les mers et on projette le nom du fleuve."* au lieu d'exprimer l'intention *"Nom des fleuves qui se jettent dans une mer"*.

Se trouver dans une situation où il y a ambiguïté : T→R

Cette situation est particulièrement complexe à gérer dans le cas général, pour l'élève comme pour l'enseignant. Il faut vraiment se trouver dans une situation où il n'y a pas d'ambiguïté. Une table résultat T est proposée et l'élève doit trouver la (une ?) requête permettant de l'obtenir. Souvent plusieurs solutions existent, et il n'est pas toujours simple de les anticiper. Devant la machine, il est possible de permettre à l'élève de vérifier sa proposition : T1→R⇒T2, T1=T2. Cette situation peut être scénarisée par un travail individuel initial sur plusieurs tables, suivi d'un débat dans la classe

permettant d'appréhender mieux la structure de la base de données et les besoins fonctionnels.

3.2 Situations didactiques utilisées dans les ouvrages et les applications

Au regard des propositions que nous avons faites, nous avons parcouru les ouvrages disponibles aux épreuves de CAPES NSI de 2021. Pour les exercices en SQL, nous constatons, comme beaucoup de sites sur le Web, que la très grande majorité des questions (90% à 100%) est de la forme I→R (Table 1). (Balabonski et al., 2020) est un peu plus original en proposant d'autres formes (30%) : R→T et R→I.

Ouvrage \ Situation	R-T	R-I	IT-R	I-R
(Connan et al., 2020)		1		9
(Bonnefoy and Petit, 2020)			1	10
(Balabonski et al., 2020)	5	4		20
(Bays, 2020)				68

TABLE 1 – Types de questions dans les ouvrages à destination des élèves

Des outils ont été proposés pour aider à la formation à SQL : SQL-Tutor (Mitrovic, 1998), SQLator (Sadiq et al., 2004), AsseSQL (Prior and Lister, 2004), SQL-KnoT (Brusilovsky et al., 2008, 2010), SQLSandbox (Desmontils, 2010), BlocklySQL (Pöhner et al., 2019) ou SQheLper (Jacobs and Jaschke, 2021) par exemple. Cependant, ces outils permettent de construire des requêtes SQL (pour certains en programmation par blocs) sur des questions qui restent sur le format "Donnez la requête qui ..." (I→R). Leur effort porte sur la construction de la requête, l'organisation de tests, l'évaluation ou la vérification des requêtes proposées par les étudiants. Le même constat peut-être fait sur une très grande majorité de cours en ligne pour NSI ou d'autres niveaux qui proposent des exercices de la forme I→R. Nous proposons donc un nouvel outil pour enseigner SQL qui couvre les différentes situations présentées.

4 L'application Apprendre-SQL

Notre objectif est de proposer une application (appelée Apprendre-SQL [3]) pour faire des exercices (ensemble de questions) plus variés qu'avec les autres outils existants. Une question est associée à une requête (éventuellement choisie dans une base de requêtes) selon une des situations didactiques vues en section 3.

Un enseignant peut installer une base de données (fichier SQL compatible avec Sqlite) mais aussi la décrire, proposer une base de requêtes SQL et constituer des exercices dont chacune des questions correspondra à une situation didactique particulière sur une requête de sa base. Dans un exercice, les questions sont présentées dans un ordre de complexité tenant compte de la situation choisie et de la complexité intrinsèque de la requête travaillée.

Quand l'élève réalise un exercice, il a, selon la situation didactique de la question, des retours instantanés. Par exemple, si la requête proposée par l'élève ne donne pas le résultat attendu (situation

3. `https://gitlab.univ-nantes.fr/ls2n-didactique/asql`

de la forme [X]Y→R), l'outil lui précise que la sortie attendue est mauvaise. A l'issu de l'exercice, deux traces sont produites. La première trace restitue toutes les actions réalisées par l'élève afin d'étudier sa manière d'aborder chaque question et donc l'apprentissage du SQL. La seconde correspond uniquement aux résultats finaux produits par l'élève que l'enseignant récupère et corrige.

5 Conclusion

Dans ce papier, nous avons présenté un ensemble de situations didactiques pour introduire le langage SQL. Ces situations permettent de développer chez l'élève des compétences liées à la manipulation d'une base de donnée. Les situations proposées sont axées sur la partie recherche du langage.

Comme pour les requêtes de recherche, il est possible de proposer des situations d'enseignement pour des modifications de la base de données. L'objectif, en plus de faire comprendre le mécanisme de ces requêtes, est de sensibiliser à la gestion de la cohérence de la base de données. Cela nécessite une bonne compréhension du modèle relationnel et des différentes contraintes, pas toujours visibles sur le modèle lui-même.

Il existe plusieurs référentiels de compétences en informatique comme dans (Selby and Woollard, 2013) ou même dans le bulletin officiel (annexe à (MENJ, 2019)). Nous avons commencé à explorer les compétences spécifiques des différentes situations présentées, mais cela reste encore à préciser. On constate en première analyse, et c'est rassurant, que le standard I→R est la modalité qui semble la plus complète. Cependant, cela montre aussi que, pour mettre en place de la progressivité, ce n'est pas forcément judicieux de commencer par cette catégorie.

Les situations didactiques présentées peuvent être mise en œuvre en NSI, mais aussi en premier cycle universitaire pour des mises à niveau, aussi bien par l'application "Apprendre-SQL" que, plus simplement, dans le cadre d'exercices de travaux dirigés sans machine. Nous avons commencé une expérimentation informelle dans cette dernière modalité en mise à niveau en 3e année de licence MIAGE (20 étudiants) sur l'algèbre relationnelle. Les premiers retours des étudiants (en particulier ceux ayant déjà eu une introduction aux bases de données) sont très positifs. D'autres expérimentations (avec et sans l'application) vont être lancées en classe de NSI et en licence à la rentrée 2023.

Références

Ahadi, A., Prior, J., Behbood, V., and Lister, R. (2015). A quantitative study of the relative difficulty for novices of writing seven different types of sql queries. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, pages 201–206.

Balabonski, T., Conchon, S., Filliâtre, J.-C., and Nguyen, K. (2020). *Numérique et sciences informatiques : 24 leçons avec exercices corrigés*. Ellipses.

Bays, S. (2020). *Spécialité Numérique et sciences informatiques*. Ellipses.

Bonnefoy, J.-C. and Petit, B. (2020). *Numérique et sciences informatiques Tle*. Ellipses.

Brusilovsky, P., Sosnovsky, S., Lee, D. H., Yudelson, M., Zadorozhny, V., and Zhou, X. (2008). An open integrated exploratorium for database courses. *AcM SIGcSE Bulletin*, 40(3) :22–26.

Brusilovsky, P., Sosnovsky, S., Yudelson, M. V., Lee, D. H., Zadorozhny, V., and Zhou, X. (2010). Learning SQL programming with interactive tools : From integration to personalization. *ACM Transactions on Computing Education (TOCE)*, 9(4) :1–15.

Chrisment, C., Pinel-Sauvagnat, K., Teste, O., and Tuffery, M. (2008). *Bases de données relationnelles : Concepts, mise en oeuvre et exercices*. Collection Informatique. Hermes Science Publications.

Connan, G., Petrov, V., Rozsavolgyi, G., and Signac, L. (2020). *Prépa bac, Spécialité NSI, Tle générale*. Hatier.

Dagiene, V., Sentance, S., and Stupuriené, G. (2017). Developing a two-dimensional categorization system for educational tasks in informatics. *Informatica*, 28(1) :23–44.

Desmontils, E. (2010). SQLSandbox. `http://www.desmontils.net/SQLSandbox/`.

Jacobs, S. and Jaschke, S. (2021). SQheLper : A block-based syntax support for SQL. In *2021 IEEE Global Engineering Education Conference (EDUCON)*, pages 478–481. IEEE.

Kearns, R., Shead, S., and Fekete, A. (1997). A teaching system for SQL. In *Proceedings of the 2nd Australasian conference on Computer science education*, pages 224–231.

Kleerekoper, A. and Schofield, A. (2018). SQL tester : an online SQL assessment tool and its impact. In *Proceedings of the 23rd annual ACM conference on innovation and technology in computer science education*, pages 87–92.

MENJ (2019). Programme de l'enseignement de spécialité de numérique et sciences informatiques de la classe terminale de la voie générale. Bulletin officiel spécial 8, Ministère de l'Éducation nationale et de la Jeunesse.

Miedema, D., Aivaloglou, E., and Fletcher, G. (2021). Identifying SQL misconceptions of novices : Findings from a think-aloud study. In *Proceedings of the 17th ACM Conference on International Computing Education Research (ICER 2021)*. ACM.

Mitrovic, A. (1998). Learning SQL with a computerized tutor. In *Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education*, pages 307–311.

Pöhner, N., Schmidt, T., Greubel, A., Hennecke, M., and Ehmann, M. (2019). BlocklySQL : A new block-based editor for SQL. In *Proceedings of the 14th Workshop in Primary and Secondary Computing Education*, pages 1–2.

Prior, J. C. and Lister, R. (2004). The backwash effect on SQL skills grading. *ACM SIGCSE Bulletin*, 36(3) :32–36.

Reisner, P. (1981). Human factors studies of database query languages : A survey and assessment. *ACM Computing Surveys (CSUR)*, 13(1) :13–31.

Renaud, K. and Van Biljon, J. (2004). Teaching SQL—which pedagogical horse for this course ? In *Key Technologies for Data Management : 21st British National Conference on Databases, BNCOD 21, Edinburgh, UK, July 7-9, 2004. Proceedings 21*, pages 244–256. Springer.

Sadiq, S., Orlowska, M., Sadiq, W., and Lin, J. (2004). SQLator : an online SQL learning workbench. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pages 223–227.

Selby, C. and Woollard, J. (2013). Computational thinking : the developing definition. University of Southampton (E-prints).

Smelcer, J. B. (1995). User errors in database query composition. *International Journal of Human-Computer Studies*, 42(4) :353–381.

Actes de l'atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université »
Brest, France, 13 juin 2023

Expérimentation du *pair-programming* en spécialité NSI en classe de première pour l'acquisition de compétences en programmation

Sophie Chane-Lune[1] Christophe Declercq[1] Sébastien Hoarau[1]
(1) Laboratoire d'informatique et de mathématiques, Université de la Réunion
Sophie.Chane-Lune@ac-reunion.fr, christophe.declercq@univ-reunion.fr,
seb.hoarau@univ-reunion.fr

RÉSUMÉ

Le *pair-programming* est une modalité d'enseignement pratique de la programmation où les deux élèves ont des rôles différents : le *pilote* programme pendant que le *co-pilote* analyse, commente, teste le travail de son binôme. L'approche par compétences permet de distinguer finement les compétences développées par les élèves et en particulier d'identifier les compétences spécifiques à chaque rôle dans une activité en *pair-programming*. En classe de première, le programme de la spécialité NSI mêle des notions d'algorithmique et de programmation avec un focus particulier sur les méthodes de programmation incluant spécification, mise au point et tests. Notre travail a consisté à expérimenter une séquence d'activités sur ces notions, pour montrer l'intérêt combiné du *pair-programming* et de l'approche par compétences.

ABSTRACT

Experimentation of pair-programming in college for the acquisition of programming skills

Pair-programming is a method of practical teaching of programming where the two students have different roles: the pilot programs while the co-pilot analyzes, comments, tests the work of his pair. The skills-based approach makes it possible to finely distinguish the skills developed by the students and in particular to identify the skills specific to each role in a pair-programming activity. In college year 12, computer science curriculum combines notions of algorithms and programming with a particular focus on programming methods including specification, development and testing. Our work consisted in experimenting with a sequence of activities on these notions, to show the combined interest of pair-programming and skills-based approach.

MOTS-CLÉS : Numérique et science informatique, compétences, collaboration, apprentissage, programmation.

KEYWORDS: Computer science education, teaching, pair-programming, skills.

1 Introduction

Au-delà du cadre de l'enseignement, le *pair-programming* est un style de programmation où deux programmeurs collaborent côte à côté sur la même machine, sur le même code, le même algorithme ou les mêmes tests (Williams & Kessler, 2003). Apparu très tôt chez les programmeurs en activité (les premiers retours datent de la fin des années 60, bien avant de s'appeler *pair-programming*), ce style de programmation fait son apparition dans le monde de l'éducation dans les années 2000 et depuis, a fait l'objet de nombreuses recherches qui montrent son intérêt, voir par exemple (Hanks et al., 2011), (Tunga & Tokel, 2018).

Néanmoins, les études concernent essentiellement le *pair-programming* dans le cadre de cours de programmation dans le supérieur et peu de choses ont été proposées pour l'apprentissage des débutants au niveau du lycée.

L'utilisation d'artefacts pour outiller le *pair-programming* a été documenté en particulier par (Schümmer & Lukosch, 2009) dans ce qu'ils nomment *distributed pair-programming*. Nous proposons, dans ce cadre, un nouvel instrument très léger permettant de mettre en œuvre à moindre coût une activité de *pair-programming* dans le cadre d'un travail pratique au lycée. Cet instrument s'apparente aussi - en version très simplifiée - aux environnements de laboratoire de travaux pratiques à distance tels que Lab4ce (Broisin et al., 2015), auquel nous empruntons l'idée de partage d'une console d'un élève avec un autre élève.

Concernant l'approche par compétences en programmation, nous nous appuyons sur les travaux fondateurs de (Wing, 2006), complétés par les précisions opérationnelles apportées par (Selby & Woollard, 2014). Nous avons aussi déjà proposé de renommer en français les compétences avec les verbes : *évaluer, anticiper, décomposer, généraliser et abstraire* (Declercq, 2022) auquel nous proposons maintenant d'ajouter la compétence *modéliser* qui nous paraît fondamentale dans le choix des structures de données étudiées en spécialité NSI.

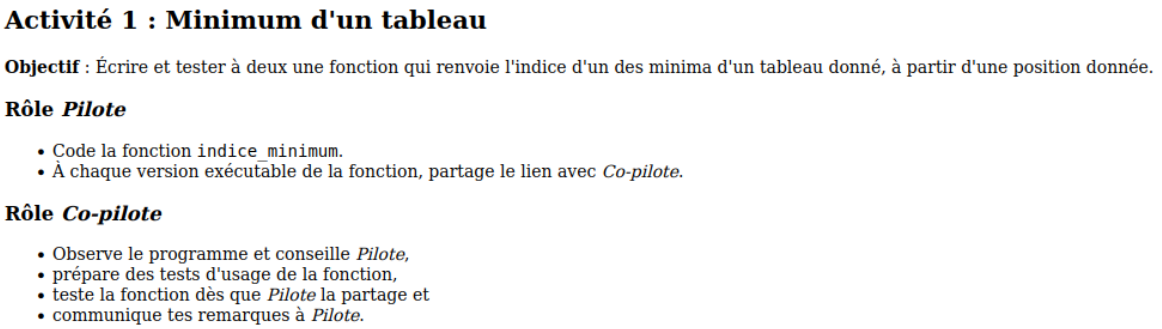
Dans cette communication, nous proposons de décrire une séquence d'activités en *pair-programming* en précisant notamment les deux rôles du binôme, ainsi que notre outil en ligne dédié à des activités de *pair-programming* ou de partage de code. Puis, nous faisons une analyse des compétences en jeu dans les activités pour chacun des rôles. Nous terminerons par une analyse de l'expérimentation réalisée en classe de première NSI et les perspectives de cette étude exploratoire.

2 Le *pair-programming*, une méthode collaborative d'apprentissage au lycée

L'idée initiale d'expérimenter cette méthode a été formulée afin de poursuivre les objectifs suivants pour les élèves :

- favoriser la réflexion en amont de l'écriture du code et de partager cette réflexion avec son binôme,
- améliorer la phase de tests des fonctions écrites,
- améliorer l'écriture de fonctions et notamment réussir à écrire des fonctions plus complexes.

Pour mettre en œuvre la méthode au lycée, l'énoncé précise aux élèves les rôles respectifs du *pilote* et du *co-pilote*. Par exemple, pour la première activité de la séance en cours d'expérimentation, l'énoncé est le suivant (voir figure 1).



Activité 1 : Minimum d'un tableau

Objectif : Écrire et tester à deux une fonction qui renvoie l'indice d'un des minima d'un tableau donné, à partir d'une position donnée.

Rôle *Pilote*

- Code la fonction `indice_minimum`.
- À chaque version exécutable de la fonction, partage le lien avec *Co-pilote*.

Rôle *Co-pilote*

- Observe le programme et conseille *Pilote*,
- prépare des tests d'usage de la fonction,
- teste la fonction dès que *Pilote* la partage et
- communique tes remarques à *Pilote*.

Figure 1: Activité 1

L'activité 2 est présentée à la section suivante (voir figure 2) dans le cadre de la présentation de notre artefact et consiste à utiliser la fonction précédente pour positionner le minimum d'un tableau en première position. La troisième activité consiste à généraliser la fonction écrite à l'activité 2 pour positionner le minimum relatif à partir d'une position donnée. La dernière activité consiste à intégrer la ou les fonctions précédentes en vue de construire le programme de tri d'un tableau par recherche du minimum.

La mise en œuvre du *pair-programming* est faite en imposant un changement de rôle à chaque activité, pour que chaque élève ait alternativement chacun des deux rôles.

3 Proposition d'un outil pour des activités en ligne de *pair-programming*

Une interface en ligne permet à l'enseignant de saisir un énoncé (en markdown), une partie de code Python fourni, puis de générer un lien pouvant être partagé avec les élèves ayant le rôle *pilote*. Les *pilotes* disposent alors, dans leur navigateur, de l'énoncé, d'un éditeur en ligne avec le code à compléter, d'une console et d'un lien permettant de partager leur travail avec leur *co-pilote* (voir figure 2).

Le *co-pilote* accède alors à l'énoncé, au code non modifiable et à une console permettant d'effectuer les tests. Le *co-pilote* ne dispose ni du bouton `Run`, ni du bouton de partage : tout ce qu'il peut faire, avec le code fourni, se déroule dans la console (voir figure 3).

Conformément aux règles du *pair-programming*, un seul élève peut modifier le code à un moment donné. Les élèves sont encouragés à échanger, le *co-pilote* ne pouvant pas modifier le code, doit l'analyser, détecter d'éventuelles erreurs par les tests et en rendre compte au *pilote*.

Les tests réalisés dans la console permettent ici de constater que le code proposé est erroné car un élément a été dupliqué et un autre perdu. La recherche, par le *co-pilote*, de l'erreur dans le code consiste à évaluer ce que le programme, fourni par le *pilote,* fait effectivement et de comparer avec ce qu'il est censé faire.

Activité 2 - Positionne le minimum dans un tableau

Cette activité est à commencer quand la fonction `indice_minimum` de l'activité 1 fonctionne (validée par *Co-pilote*). Commencer par échanger les rôles (*Pilote* ⇔ *Co-pilote*).

Objectif

Écrire et tester à deux une fonction `positionne_minimum` qui prend un tableau en paramètre et qui met en première position du tableau, la plus petite valeur de ce tableau. **Attention**, la fonction modifie bien le contenu du tableau, mais on ne doit perdre **aucune** des valeurs.

```
def positionne_minimum (tableau):
    tableau[0] = 0
```

RUN

```
Brython 3.11.2 on Netscape 5.0 (X11; Ubuntu)
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Partage la console | Partage éditeur et console

Figure 2: Vue du pilote

Activité 2 - Positionne le minimum dans un tableau

Cette activité est à commencer quand la fonction `indice_minimum` de l'activité 1 fonctionne (validée par *Co-pilote*). Commencer par échanger les rôles (*Pilote* ⇔ *Co-pilote*).

Objectif

Écrire et tester à deux une fonction `positionne_minimum` qui prend un tableau en paramètre et qui met en première position du tableau, la plus petite valeur de ce tableau. **Attention**, la fonction modifie bien le contenu du tableau, mais on ne doit perdre **aucune** des valeurs.


```
def indice_minimum(tableau):
    min, i_min = tableau[0], 0
    for i in range (len(tableau)):
        if tableau[i] < min:
            min, i_min = tableau[i], i
    return i_min
def positionne_minimum (tableau):
    i = indice_minimum(tableau)
    tableau[0] = tableau[i]
```

```
Brython 3.11.2
>>> valeurs = [92, 45, 89, 34, 56, 26, 38]

>>> positionne_minimum(valeurs)
>>> valeurs
[26, 45, 89, 34, 56, 26, 38]
>>> positionne_minimum(valeurs)
>>> valeurs
[26, 45, 89, 34, 56, 26, 38]
>>> valeurs = [92, 45, 89, 34, 56, 26, 38]


>>> positionne_minimum(valeurs)
>>> valeurs
[26, 45, 89, 34, 56, 26, 38]
>>>
```

Figure 3: Vue du co-pilote

4 Approche par compétences de l'apprentissage de la programmation

Les programmes de spécialité NSI déclinent une liste de capacités attendues dans un référentiel organisé par domaines et notions. La granularité de ces capacités est assez fine ; certaines sont liées entre elles, en particulier entre les domaines programmation et algorithmique. De manière assez orthogonale, les travaux dans le cadre de *computational thinking* sur les compétences acquises à l'occasion de l'apprentissage de la programmation, identifient des compétences cognitives très générales. Nous tentons actuellement de croiser les deux approches. Le tableau suivant donne un extrait des compétences en programmation que nous avons identifiées et catégorisées. Cet extrait concerne le domaine des tableaux incluant construction, parcours et tri de ces tableaux.

Tableau 1: Compétences en programmation - extrait concernant les tableaux

	Construire un tableau et accéder à ses éléments	***Parcourir un tableau pour rechercher ou calculer des informations***	***Parcourir et modifier un tableau par des boucles imbriquées***
Évaluer	E1 : évaluer le résultat d'un programme utilisant un tableau	E2 : évaluer le résultat d'un programme itérant sur un tableau	E3 : évaluer les valeurs successives des indices avec des boucles imbriquées
Modéliser	M1 : modéliser une série d'informations par un tableau donné en extension	M2 : modéliser un traitement de tableau en choisissant d'itérer sur les indices ou les éléments	
Anticiper	A1 : anticiper les traitements à programmer pour accéder aux éléments d'un tableau par leurs indices et les modifier	A2 : anticiper un traitement nécessitant un parcours linéaire d'un tableau	A3 : anticiper un traitement de tableau nécessitant plusieurs parcours imbriqués
Décomposer			D3 : décomposer en identifiant des parcours séquentiels de tableaux
Généraliser	G1 : généraliser l'écriture d'un tableau en utilisant la compréhension		
Abstraire	X1 : abstraire les données en considérant que les éléments d'un tableau peuvent être des tableaux		X3 : abstraire un parcours séquentiel en utilisant une fonction avec un tableau en paramètre

Les compétences que nous avons retenues articulent savoirs et savoirs-faire utilisés en contexte.

4.1 Analyse a priori des compétences développées selon les rôles

Lors des activités 1, 2 et 3, on attend du *pilote* qu'il développe les compétences A1 et A2. Il est en effet en charge de l'écriture de programmes nécessitant d'itérer sur un tableau.

Le *co-pilote* lui, va apprendre à évaluer les programmes fournis, et développer ainsi les compétences E1 et E2. Il aura aussi à modéliser les données : compétence M1.

Lors de la quatrième activité (tri par recherche du minimum), le *pilote* va développer la compétence A3. Les compétences D3 et X3 ont, quant à elles, été prises en charge par l'enseignant qui a proposé un énoncé guidé et a ainsi choisi la décomposition du problème et en a proposé une abstraction.

Le *co-pilote* développera pour sa part, la compétence E3. Une observation fine des activités des élèves pourra aussi mettre en évidence la maîtrise de la compétence G1 (écriture de tableaux en compréhension).

En résumé, sur l'ensemble de l'activité, ce sont principalement les compétences d'anticipation qui vous pouvoir être développées par le *pilote* et celles d'évaluation par le *co-pilote*. L'échange de rôles institutionnalisé par le *pair-programming*, est ainsi dans ce contexte un moyen d'amener les élèves à développer toutes les compétences.

On en déduit aussi qu'il n'y a pas lieu d'inscrire à notre référentiel de compétence spécifique au test. L'activité de test est une activité transversale qui gagne à être pratiquée tout au long des activités de programmation. Les compétences développées au cours des tests sont spécifiquement les compétences d'évaluation ainsi que celles de compréhension des modélisations fournies par les spécifications.

5 Expérimentation en classe de première NSI et premiers résultats

La mise au point de cette expérimentation a été effectuée au cours du premier trimestre 2023. La séquence a été testée en classe le 17 avril 2023 avec une classe de première NSI du lycée Le Verger à Sainte-Marie (La Réunion).

La classe comporte 18 élèves dont 16 garçons et 2 filles. La progression annuelle de cette classe a permis d'introduire les tableaux et les parcours simples au premier trimestre. Les activités 1 à 3 consistent donc en un réinvestissement de notions déjà abordées, alors que l'activité 4 a permis l'introduction au tri d'un tableau. Ce choix a été fait pour simplifier la genèse instrumentale avec un artefact nouveau, assez différent des notebooks avec lesquels les élèves avaient pris l'habitude de travailler en Python.

Le dispositif de recueil de données a permis de récolter les traces d'activités de 6 binômes. Chaque trace (en JSON) comporte l'horodatage de toutes les actions du *pilote* dans l'éditeur (charger, exécuter, partager) et du *co-pilote* dans sa console (charger, enregistrer). Par exemple, la figure 4 montre un extrait de la trace d'activité d'un *co-pilote,* au moment où il signale une erreur de programmation lors de l'activité 2 : « Si on lance le programme 2 fois avec le même tableau, ça fait une erreur », erreur provoquée par l'oubli d'initialisation de la variable indice.

```
2023-04-17T10:55:14.121000
MODE    : co-pilote
ACTION  : charger
EDITEUR :
def positionne_minimum(valeurs):
    mini=valeurs[0]
    for i in range(len(valeurs)):
        if mini>valeurs[i]:
            indice=i
            mini=valeurs[i]
    nb=valeurs[0]
    valeurs[0]=mini
    valeurs[indice]=nb
    return valeurs

2023-04-17T10:57:56.797000
MODE    : co-pilote
ACTION  : charger
EDITEUR :
def positionne_minimum(valeurs):
    mini=valeurs[0]
    indice=0
    for i in range(len(valeurs)):
        if mini>valeurs[i]:
            indice=i
            mini=valeurs[i]
    nb=valeurs[0]
    valeurs[0]=mini
    valeurs[indice]=nb
    return valeurs

2023-04-17T11:01:20.078000
MODE    : co-pilote
ACTION  : enregistrer
EDITEUR :
def positionne_minimum(valeurs):
    mini=valeurs[0]
    indice=0
    for i in range(len(valeurs)):
        if mini>valeurs[i]:
            indice=i
            mini=valeurs[i]
    nb=valeurs[0]
    valeurs[0]=mini
    valeurs[indice]=nb
    return valeurs
CONSOLE :
Brython 3.11.2
>>> valeurs = [40, 50, 10, 20, 30]
>>> positionne_minimum(valeurs)
[10, 50, 40, 20, 30]
>>> positionne_minimum(valeurs)
[10, 50, 40, 20, 30]
```

Figure 4 : Extrait de la trace d'activité du co-pilote du binôme 3.

On constate que les tests effectués après correction de la fonction permettent de vérifier que cette erreur a bien été corrigée.

Le tableau 2 résume les indicateurs calculés à partir de l'analyse des logs via un programme Python développé pour l'occasion. Nous avons en particulier calculé le nombre de partages de code par activité pour chaque binôme, et testé les programmes partagés avec des jeux de tests dédiés pour en déduire la réussite de chaque binôme à chaque activité.

Tableau 2: Indicateurs d'activité et de réussite des binômes par activité.

Binôme\Act	Activité 1 Partages	Activité 1 Réussite	Activité 2 Partages	Activité 2 Réussite	Activité 3 Partages	Activité 3 Réussite	Activité 4 Partages	Activité 4 Réussite
1	2	oui	1	oui	1	oui	1	oui
3	5	oui	3	oui	1	oui	1	oui
4	1	oui	2	oui	0	non	0	non
6	3	oui	3	oui	3	oui	2	oui
8	2	oui	1	oui	1	oui	1	oui
9	5	oui	3	oui	0	non	0	non

L'analyse globale des traces d'activités a ainsi permis de mettre en évidence les interactions entre *pilotes* et *co-pilotes* (attesté par le nombre de partages de code pour chaque activité), ainsi qu'une réussite globale à l'activité supérieure à ce qui pouvait être constaté habituellement par l'enseignante. Vu le faible nombre d'élèves, ces résultats n'ont bien sûr pas de valeur statistique, mais nous semblent cependant encourageants par rapport à l'intérêt de la méthode.

6 Conclusion

Ce travail en cours explore une méthodologie d'apprentissage de la programmation - le *pair-programming* - et propose un artefact prototype adapté en tant qu'objet d'étude. L'expérimentation a aussi permis d'améliorer la méthodologie de recueil de traces d'activités.

Il utilise par ailleurs le travail - en cours aussi - d'élaboration d'un référentiel de compétences en programmation adapté à la spécialité NSI.

Nous espérons ainsi contribuer à une meilleure compréhension des difficultés rencontrées par les élèves lors de leur apprentissage, en identifiant finement les compétences en jeu et en proposant des instruments incitant à diversifier les activités et amenant les élèves à collaborer.

Parmi les perspectives de cette étude exploratoire, nous envisageons d'une part une étude qualitative plus longue expérimentant de manière systématique le *pair-programming* sur une durée beaucoup plus longue dans une classe, et d'autre part une étude quantitative de l'appropriation de l'artefact avec un plus grand nombre d'élèves ou étudiants.

Références

Broisin, J., Venant, R., & Vidal, P. (2015). Lab4CE: a Remote Laboratory for Computer Education. *International Journal of Artificial Intelligence in Education*, *25*(4), 154-180.

Declercq, C. (2022). *Didactique de l'informatique : Une formation nécessaire*. Sticef *28*. http://sticef.org/num/vol2021/28.3.8.declercq/28.3.8.declercq.htm

Hanks, B., Fitzgerald, S., McCauley, R., Murphy, L., & Zander, C. (2011). Pair programming in education : A literature review. *Computer Science Education*, *21*(2), 135-173. https://doi.org/10.1080/08993408.2011.579808

Schümmer, T., & Lukosch, S. G. (2009). Understanding Tools and Practices for Distributed Pair Programming. *J.UCS (Annual Print and CD-ROM Archive Ed.)*. https://www.narcis.nl/publication/RecordID/oai:tudelft.nl:uuid:9a29882c-8a6b-441b-b83b-e61d5a0b448b

Selby, C., & Woollard, J. (2014). Computational thinking : The developing definition. *SIGCSE*.

Tunga, Y., & Tokel, T. (2018). *The use of pair programming in education : A systematic literature review*.

Williams, L., & Kessler, R. (2003). *Pair programming illuminated*. Addison-Wesley.

Wing, J. M. (2006). Computational Thinking. *Communications of the ACM*, *49*.

Actes de l'atelier « Apprendre la Pensée Informatique de la Maternelle à l'Université »
Brest, France, 13 juin 2023

SPY : Un jeu sérieux partagé pour étudier l'apprentissage de la pensée informatique

Mathieu Muratet[1, 2]
(1) Sorbonne Université, CNRS, LIP6, F-75005 Paris, France
(2) INSHEA, 92150 Suresnes, France
mathieu.muratet@lip6.fr

RÉSUMÉ

Dans cet article nous présentons le jeu sérieux SPY comme un EIAH partagé à la fois pour son usage, son code source et ses traces d'interaction. SPY est un jeu sérieux *open source* sur le thème de la pensée informatique. Nous présentons les scénarios actuellement inclus dans le jeu comme une illustration des champs des possibles. Ces scénarios sont utilisables en l'état par des enseignants qui souhaiteraient faire travailler leurs élèves sur les compétences de la pensée informatique. Nous montrons comment le jeu peut être étendu à travers la création de nouveaux scénarios et niveaux. Enfin nous partageons les données produites par le jeu au format xAPI afin de créer une base de données ouverte permettant aux chercheurs de la communauté de les analyser.

ABSTRACT

SPY: a shared serious game to study computational thinking.

In this paper we present the serious game SPY as a shared ITS for its use, its source code and its traces. SPY is an open source serious game on computational thinking. We present the scenarios currently included into the game as an illustration of the scope of possibilities. These scenarios can be used by teachers who would like to make their students work on computational thinking. We show how the game can be extended through the creation of new scenarios and levels. Finally, we share the data produced by the game in xAPI format in order to create an open database available to researchers.

MOTS-CLÉS : jeu sérieux, pensée informatique, données ouvertes, partage.
KEYWORDS: serious game, computational thinking, open data, sharing.

1 Introduction

Depuis maintenant quelques années, l'informatique disciplinaire réapparaît dans les programmes scolaires (Baron et Drot-Delange, 2016), on parle de pensée informatique. Selon Wing (Wing, 2006), la pensée informatique met en jeu un répertoire de cinq capacités cognitives : (1) la pensée algorithmique, (2) l'abstraction, (3) l'évaluation, (4) la décomposition, et (5) la généralisation. L'appropriation de ces compétences par les enseignants de l'école primaire est complexe à résoudre. En effet, enseigner cette nouvelle discipline demande un investissement particulièrement important de leur part (Kradolfer *et al.*, 2014) notamment en raison d'un manque de formation (initiale et continue). Ainsi, par manque de maîtrise de ces compétences les enseignants se trouvent en

difficulté pour construire leurs propres scénarios pédagogiques ou juger de la pertinence d'outils pour faire travailler ces compétences à leurs élèves (Roche *et al.*, 2018).

De nombreuses ressources permettant de travailler les compétences de la pensée informatique existent. Certaines mettent en avant l'informatique débranchée qui est un point d'entrée de la science informatique riche et qui présente l'avantage de ne pas nécessiter de matériels coûteux (Alayrangues *et al.*, 2017). Ces mêmes auteurs notent que mener de telles activités requiert des connaissances en informatique, et des compétences en pédagogie. Ce « super prof » maîtrisant ces deux facettes reste une exception, le passage à l'échelle avec ce type de scénario pédagogique reste donc difficile.

D'autres ressources exploitent des robots programmables. Ici l'objet tangible fournit un artefact fertile sur le plan cognitif pour développer des compétences mathématiques et des formes de pensées algorithmiques (Komis et Misirli, 2011). Ces robots programmables sont alors exploités dans le cadre de scénarios pédagogiques. L'enseignant doit alors maîtriser les concepts sous-jacent pour soutenir les travaux des élèves. Michel Spach (Spach, 2019) souligne là encore, dans le contexte de la robotique pédagogique, les freins au développement de ces ressources : « Le défaut de maîtrise conceptuelle des enseignants est sans doute à l'origine du manque de référencement et du déficit d'institutionnalisation des notions et des concepts abordés dans les situations pédagogiques ».

Enfin, de nombreuses ressources numériques et ludiques complètent ce panorama (Miljanovic et Bradbury, 2018 ; Vahldick *et al.*, 2014). Ces EIAH (Environnement Informatique pour l'Apprentissage Humain) couvrent un spectre large allant de l'enseignement primaire à l'université ; de l'initiation à la programmation avec des langages à base de blocs à la programmation d'intelligences artificielles avec des langages avancés. L'offre est donc riche et consistante même pour les élèves de l'école élémentaire, pour autant les ressources existantes restent difficilement adaptables (Saddoug *et al.*, 2022), d'une part car la plus part de ces EIAH ne fournissent tout simplement pas d'outils informatiques permettant de modifier les scénarisations existantes ou d'en créer de nouvelles et, d'autre part, pour les rares qui le permettent comme Kodu, Scratch ou Code.org, ils ne proposent pas d'aide à l'enseignant pour comprendre les situations créées comme, par exemple, expliciter les compétences en jeu.

Les ressources numériques présentent un certain nombre d'avantage par rapport aux approches débranchées ou robotiques. D'une part, d'un point de vue de l'usage elles ne demandent pas l'achat de ressources spécifiques comme c'est le cas pour les robots de sol et peuvent embarquer des aides à la fois à destination des apprenants et des enseignants. D'autre part, d'un point de vue recherche en informatique elles présentent l'avantage de pouvoir générer des traces d'interactions qui peuvent être étudiées pour analyser les productions et les comportements des apprenants mais aussi des enseignants.

Cet article a pour objectif de présenter et partager un EIAH particulier sur le thème de la pensée informatique, il s'agit du jeu sérieux SPY que nous présenterons dans la section 2. Ce jeu sérieux est présenté dans une perspective d'ouverture vis à vis de la communauté (section 3) afin de proposer une ressource libre d'utilisation, modifiable et partagée (à la fois du point de vue de son code source et des données produites). Enfin nous conclurons cet article en proposant quelques pistes de recherches.

2 SPY : le jeu

SPY est un jeu sérieux d'apprentissage sur le thème de la pensée informatique. Il a été conçu pour un public d'élèves de cycle 3 (CM1, CM2, 6ème). Le principe du jeu est de programmer un ou plusieurs agents à l'aide d'actions simples afin de les déplacer sur une grille vers des positions particulières. Ces actions sont représentées sous forme de blocs que le joueur doit agencer en séquences.

Pour programmer le robot, le joueur bénéficie de blocs d'action (« Avancer », « Pivoter à gauche », « Pivoter à droite », « Attendre », « Activer un terminal » et « Faire demi-tour »), de blocs de contrôle (« Si Alors », « Si Alors Sinon », « Répéter n fois », « Tant que » et « Répéter indéfiniment »), de capteurs (« Mur en face », « Mur à gauche », « Mur à droite », « Passage en face », « Passage à gauche », « Passage à droite », « Ennemi en face », « Zone surveillée en face », « Porte détectée en face », « Terminal détecté sur ma position » et « Sortie détectée sur ma position ») et d'opérateurs (« Et », « Ou » et « Non »).

La figure 1 présente le tutoriel du jeu, dans cette scène le joueur doit faire avancer le robot d'une seule case et dispose pour cela d'une seule action « Avancer » qu'il doit glisser dans la zone d'édition de « Karl » (le nom du robot). La figure 2 présente un niveau avancé du scénario « Sélectionneur » (avec la solution) demandant au joueur de programmer deux robots avec une seule ligne de programme, le joueur devra alors trouver une solution générale permettant de résoudre deux problèmes proches mais différents. Le joueur devra alors manipuler des structures de contrôle, des capteurs et des opérateurs. La solution incluse dans la figure 2 peut être traduite ainsi « tant que je ne suis pas sur la sortie, avancer d'une case et si un mur est détecté sur la prochaine case en face alors pivoter à gauche (fin tant que) ».

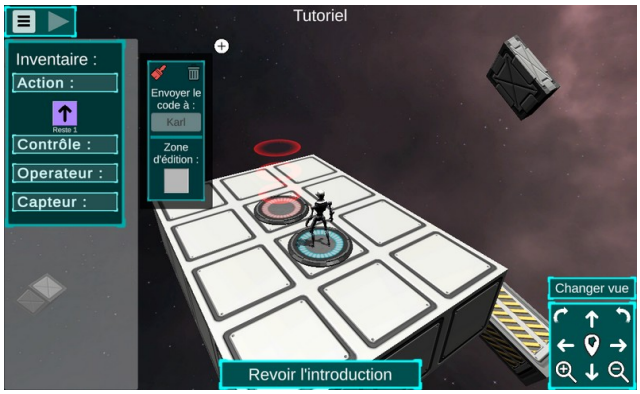


Figure 1: Tutoriel du jeu SPY

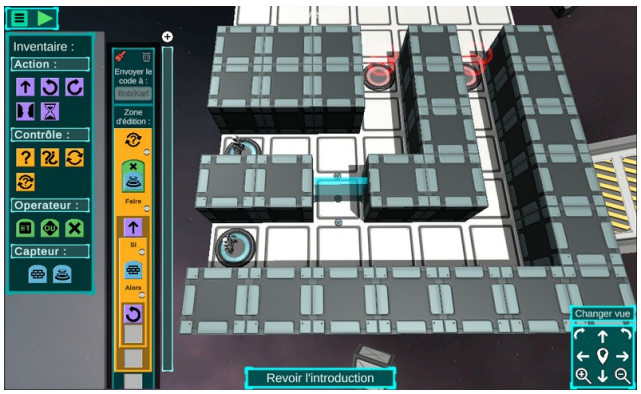


Figure 2: Un niveau du scénario « Sélectionneur »

Au moment de l'écriture de cet article, SPY contient 6 scénarios. Le premier intitulé « Explorateur » est composé de 9 niveaux et ne demande au joueur de ne manipuler que des blocs d'action. Les 4 premiers niveaux donnent seulement accès au blocs requis, le joueur doit donc les agencer dans le bon ordre sans avoir à se demander si tel ou tel bloc est utile ou non. Les 5 derniers niveaux fournissent les blocs d'action en quantité illimitée, c'est alors au joueur de déterminer quels blocs doivent être utilisés et dans quel ordre. Pour chacun de ces niveaux le joueur peut décomposer son programme en sous parties, il peut déposer quelques blocs, exécuter son programme, observer la nouvelle situation et poursuivre sa résolution sans avoir à recommencer du début.

Le deuxième scénario intitulé « Collaborateur » est composé de 12 niveaux. Dans ce scénario le joueur devra faire collaborer deux robots. Il devra notamment apprendre à nommer correctement les zones de programmation afin de contrôler les bons robots. Dans ce scénario 3 niveaux ne demandent pas au joueur de programmer mais proposent des solutions pré-construites, le joueur doit alors lire ces programmes, les comprendre et choisir celui qu'il souhaite envoyer au robot. Deux niveaux contiennent des solutions erronées que le joueur doit corriger.

Le troisième scénario intitulé « Répétiteur » est composé de 10 niveaux. Ce scénario est centré sur la notion de répétition. Ici les blocs actions sont limités non pas pour réduire la charge cognitive du joueur mais pour forcer l'utilisation du bloc « Répéter n fois ». Ce scénario exploite aussi les agents ennemis qui contiennent des actions que le joueur peut consulter mais ne peut pas modifier. Le joueur doit alors anticiper les actions de l'ennemi et programmer son robot en conséquence.

Le quatrième scénario intitulé « Sélectionneur » est composé de 8 niveaux. Ce scénario permet de découvrir les structures de contrôle « Si Alors » et « Tant que » ainsi que quelques capteurs et opérateurs. Deux niveaux de ce scénario cachent la position à atteindre, le joueur ne peut donc pas savoir où déplacer son robot, l'algorithme de résolution est donc donné dans la consigne en langage naturel ou sous la forme d'un algorigramme. Le joueur doit alors traduire cette consigne à l'aide du langage formel du jeu.

Le cinquième scénario intitulé « Infiltration » est composé de 22 niveaux. Il s'agit d'un scénario de synthèse qui reprend l'ensemble des compétences travaillées dans les 4 premiers scénarios. Dans ce scénario, les niveaux sont liés les uns aux autres au travers d'une histoire originale.

Le sixième scénario intitulé « BlocklyMaze » est une reproduction du jeu originale du même nom[1]. L'objectif ici était de vérifier qu'il était possible de décrire les niveaux d'un autre jeu avec le modèle de niveau de SPY. En effet, l'ensemble des niveaux de SPY sont décrit dans des fichiers XML extérieurs au jeu, il est ainsi possible de créer ses propres niveaux de jeu et de les agencer sous la forme de scénarios.

Cette présentation succincte du jeu et de ces scénarios a pour objectif de donner un aperçu du champ des possibles qui ne demande qu'à être étendu par la création de nouveaux niveaux, de nouveaux scénarios, de nouvelles fonctionnalités, de nouvelles analyses de données. Dans la section suivante nous présentons les différentes composantes du jeu partagé.

3 SPY : l'objet partagé

SPY est tout d'abord partagé en tant que ressource libre d'utilisation. Le jeu est accessible en ligne[2] sous la forme d'une application WebGL fonctionnant sur navigateur. L'application fonctionne sur ordinateur et sur tablette. Avec ces 6 scénarios par défaut, cette ressource peut être utilisée telle quelle par les enseignants. Pour aider les enseignants à appréhender les compétences travaillées dans chacun des scénarios, une analyse automatique des niveaux composant le scénario sélectionné est présentée (voir figure 3). Actuellement trois référentiels sont intégrés au jeu (le PIAF (Parmentier *et al.*, 2020), le CRCN[3] et celui spécifique à SPY), l'enseignant peut ainsi avoir un aperçu des compétences travaillées dans chacun des scénarios (Muratet, 2023).

1 https://blockly.games/maze, accédé le 24/05/2023
2 https://spy.lip6.fr/, accédé le 24/05/2023
3 https://eduscol.education.fr/document/20389/download, accédé le 24/05/2023

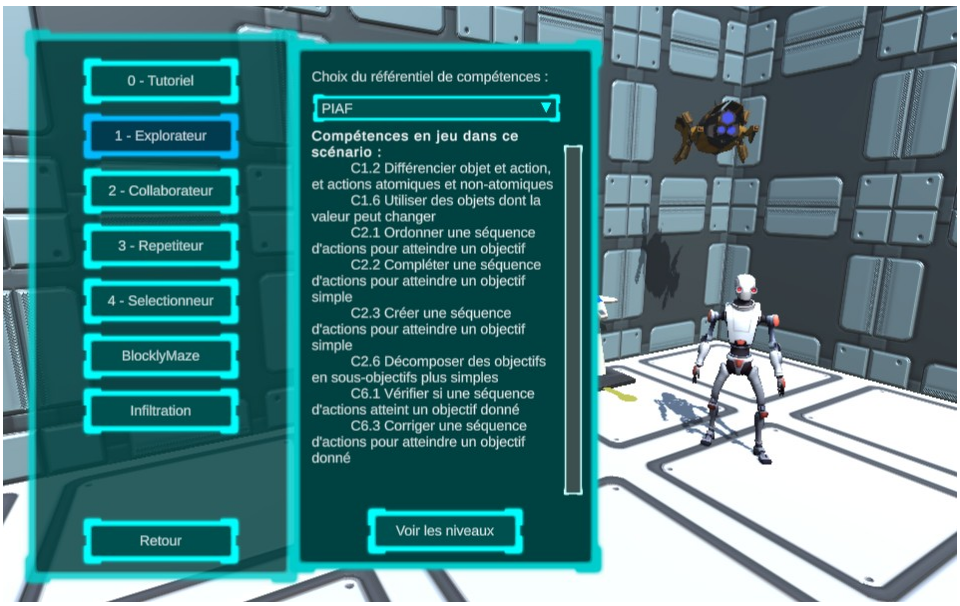


Figure 3: Analyse des compétences issues du référentiel PIAF dans le scénario « Explorateur »

L'éditeur de scénario (voir figure 4) inclus dans le jeu permet un premier niveau d'adaptation. Cet éditeur permet de filtrer la base de données des niveaux de SPY par compétences, de consulter chaque niveau en visualisant notamment les compétences en jeu et d'agencer les niveaux retenus en séquence. Un enseignant peut ainsi créer son propre scénario ou modifier un scénario existant en le (re)composant à partir des niveaux existants. La consigne de chaque niveau du scénario peut être redéfinie en vue d'adapter son contenu à un nouveau profil d'élève ou pour lier les niveaux les uns aux autres sous la forme d'une histoire.

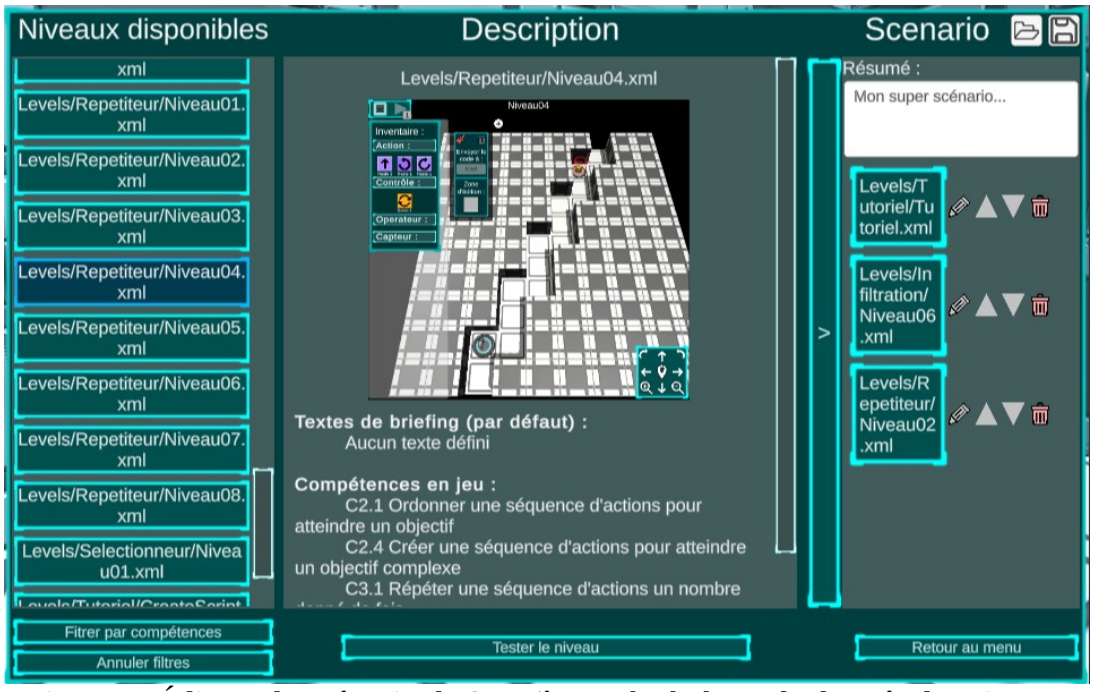


Figure 4: Éditeur de scénario de SPY (à gauche la base de donnée des niveaux du jeu, au centre le niveau actuellement sélectionné, à droite les niveaux constituant le scénario en cours d'édition)

SPY est aussi partagé en tant que projet open source[4]. Le jeu est développé par Sorbonne Université avec l'environnement Unity[5]. Nous ne rentrons pas ici sur les détails d'implémentation du jeu et nous présenterons simplement un résumé du modèle de donnée décrivant un niveau[6]. L'objet « map » décrit la topologie du niveau sous la forme d'une matrice. Les valeurs de la matrice codent la nature du sol (mur invisible, sol, mur, point de départ, point d'arrivée). L'objet « dialogs » décrit les informations affichées au chargement du niveau, il permet par exemple de présenter une consigne. Le contenu d'un dialogue peut être un texte, une image, un son, une vidéo ou une combinaison de tous ces éléments. L'objet « blockLimits » décrit les blocs disponibles dans l'inventaire du joueur, chaque type de bloc peut être présent en quantité illimitée, limitée ou non disponible. Ceci permet de réduire la complexité d'un niveau en ne proposant que les blocs utiles à sa résolution ou au contraire forcer l'utilisation de certains blocs. Les objets « player » et « enemy » permettent de positionner respectivement sur la carte des robots contrôlés par le joueur et des ennemis observant une partie de la carte. Chaque agent (robot et ennemi) écoute sur une ligne de communication. L'objet « script » permet de définir une zone de programme qui enverra son contenu sur une ligne de communication. Si un robot ou un ennemi écoute sur cette ligne alors il sera commandé par cette zone de programme. Une zone de programme peut être vide ou pré-remplie. D'autres objets peuvent être définis pour par exemple positionner des portes et des terminaux de contrôle, bloquer le *drag&drop*, limiter le nombre d'exécution... Bien que le jeu ne contienne pas pour l'instant d'éditeur de niveaux, il reste néanmoins possible de créer de nouveaux niveaux en éditant manuellement les fichiers XML.

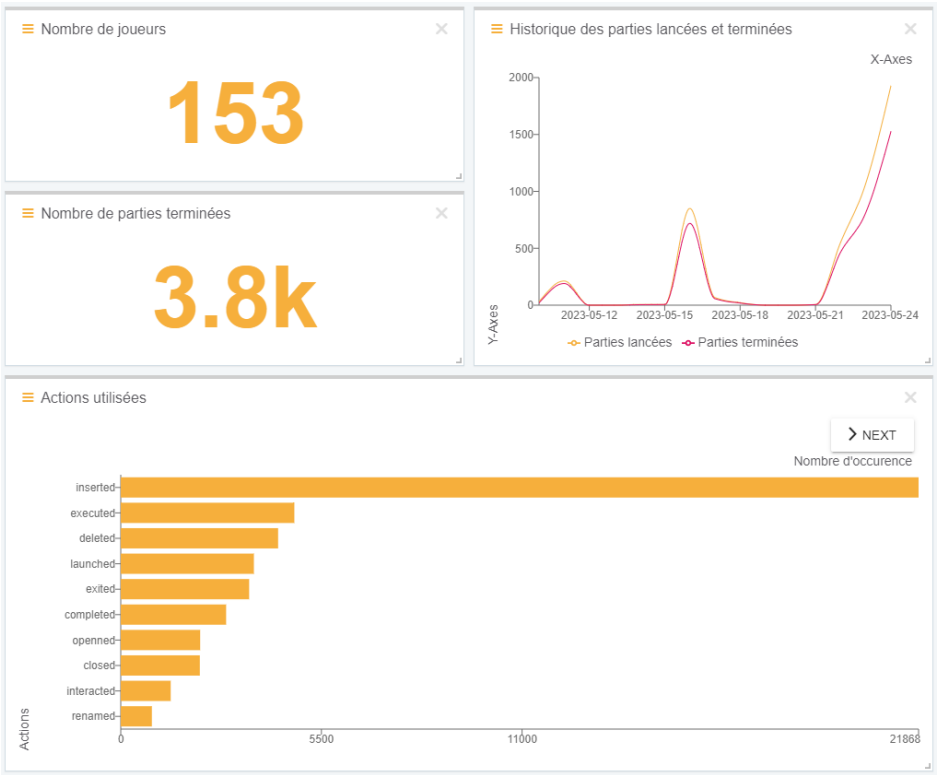


Figure 5: Exemple de visualisation générée par LearningLocker.

4 https://github.com/Mocahteam/SPY, accédé le 24/05/2023
5 https://unity.com/, accédé le 24/05/2023
6 https://github.com/Mocahteam/SPY/blob/master/Doc/LevelModel.xml, accédé le 24/05/2023

SPY est enfin partagé du point de vue des données d'interactions générées par le jeu. L'ensemble des interactions sont tracées au format xAPI[7] et stockées sur un LRS (Learning Record Store). Les données sont accessibles via un LearningLocker[8]. L'outil LearningLocker permet de générer des visualisations comme illustrées dans la figure 5. Chaque trace au format xAPI est appelé *statement* et se compose d'un triplet Acteur/Verbe/Objet (Qui ? Fait quoi ? Sur quoi ?). Chaque nouvelle connexion au jeu génère un identifiant de session anonyme unique que le joueur peut noter en vue de recharger sa progression ultérieurement. Cet identifiant de session définit l'acteur du *statement*. Les verbes et les objets permettent de caractériser le contenu des interactions. Les interactions actuellement tracées sont les suivantes : lancement du niveau, ouverture des dialogues, passage au dialogue suivant, passage au dialogue précédent, fermeture des dialogues, création d'une zone de programme, nettoyage d'une zone de programme, suppression d'une zone de programme, renommage d'une zone de programme, insertion d'un bloc, suppression d'un bloc, modification d'un bloc, exécution d'un programme, mise en pause de l'exécution, exécution pas à pas, arrêt de l'exécution d'un programme, fin du niveau, fermeture du niveau. Certains *statements* contiennent des extensions qui permettent de stoker des informations complémentaires comme par exemple les conditions de fin du niveau (succès/échec, score) ou le contexte dans lequel un bloc a été ajouté/supprimé. Actuellement seule la partie de SPY manipulée par l'élève est tracée mais nous envisageons de tracer également l'éditeur de scénario et s'il existe un jour, l'éditeur de niveau.

4 Conclusion

Dans cet article nous avons présenté le jeu SPY comme un EIAH partagé à la fois pour son usage, son code source et ses traces d'interaction. Nous avons décrit les scénarios actuellement présents dans le jeu comme une illustration des champs des possibles. Ces scénarios sont utilisables en l'état par des enseignants qui souhaiteraient faire travailler leurs élèves sur les compétences de la pensée informatique. Nous avons vu qu'il était possible d'augmenter le jeu en créant de nouveaux scénarios à partir de la base de données des niveaux existants et de créer/modifier des niveaux par l'édition de fichiers XML. Cette fonctionnalité est peu accessible mais, à défaut d'un éditeur, donne tout de même la possibilité de créer et de tester de nouvelles situations. Enfin nous proposons de partager aussi les données produites par le jeu au format xAPI afin de créer une base de données ouverte permettant aux chercheurs de la communauté de les analyser.

Nous terminerons cet article par un dernier partage, celui des questions de recherche. Nous esquissons ainsi quelques problématiques et perspectives de travaux qui pourraient être menés avec cette ressource. Comme nous l'avons présenté en introduction, un des enjeux dans ce champ de l'enseignement de la pensée informatique est d'outiller les enseignants (particulièrement de l'école primaire) avec des ressources qu'ils comprennent et qu'ils peuvent s'approprier. Quelles informations (compétences, conseils, recommandations) peuvent être extraites de la description des niveaux pour aider l'enseignant à s'approprier le jeu ? Comment les enseignants transforment-ils une telle ressource pour l'adapter à leurs élèves ? Que peut-on comprendre du processus de raisonnement des élèves à travers leurs traces d'interaction ? Comment produire des *feedbacks* à destination de l'élève pour l'aider lui et indirectement l'enseignant à dépasser les difficultés rencontrées ? ... Voici donc quelques questionnements qui, sans être exhaustifs, peuvent être abordés avec cette ressource partagée.

---

7 https://adlnet.gov/projects/xapi/, accédé le 24/05/2023

8 Toutes les informations permettant d'accéder aux traces du jeu sont disponibles à l'adresse suivante : https://spy.lip6.fr/openTraces.html, accédé le 24/05/2023

Références

Alayrangues, S., Peltier, S. and Signac, L. (2017). Informatique débranchée : construire sa pensée informatique sans ordinateur. *Colloque Mathématiques en Cycle 3 IREM de Poitiers*, IREM de Poitiers, Poitiers, France. pages 216-226.

Baron, G-L. and Drot-Delange, B. (2016). L'informatique comme objet d'enseignement à l'école primaire française ? Mise en perspective historique. *Revue française de pédagogie Recherches en éducation*, pages 51–62.

Komis, V. and Misirli, A. (2011). Robotique pédagogique et concepts préliminaires de la programmation à l'école maternelle : une étude de cas basée sur le jouet programmable BeeBot. *Didapro 4 – DidaSTIC*, Patras (Grèce).

Kradolfer, S., Dubois, S., Riedo, F., Mondada, F. and Fassa, F. (2014). A sociological contribution to understanding the use of robots in schools: the thymio robot. In *International Conference on Social Robotics*, Springer, Cham, pages 217–228.

Miljanovic, M.A. and Bradbury, J.S. (2018). A Review of Serious Games for Programming. In: *Joint International Conference on Serious Games*. Lecture Notes in Computer Science, volume 11243, pages 204-216. Springer, Cham.

Muratet, M. (2023). Comment caractériser et analyser les compétences de la pensée informatique d'un jeu sérieux ? *11ème Conférence sur les Environnements Informatiques pour l'Apprentissage Humain (EIAH 2023)*, 12 - 16 juin 2023, Brest, France.

Parmentier, Y., Reuter, R., Higuet, S., Kataja, L., Kreis, Y., Duflot-Kremer, M., Laduron, C., Meyers, C., Busana, G., Weinberger, A. and Denis, B. (2020). PIAF: developing computational and algorithmic thinking in fundamental education. In: *EdMedia+ Innovate Learning, Association for the Advancement of Computing in Education (AACE)*, pages 315–322.

Roche, M., de La Higuera, C. and Michaut, C. (2018). Enseigner la programmation informatique : comment réagissent les professeurs des écoles ? *Notes du CREN*, n°27, pages 1-8.

Saddoug, H., Rahimian, A., Marne, B., Muratet, M. and Sehaba, K. (2022). Review of the Adaptability of a Set of Learning Games Meant for Teaching Computational Thinking or Programming in France. *Special Session on Gamification on Computer Programming Learning*, Prague, Czech Republic. Pages 562-569.

Spach, M. (2019). Activités robotiques à l'école : approches de pratiques d'enseignement et effets sur les apprentissages. *Recherches en didactiques*, volume 28, pages 68-87.

Vahldick, A., Mendes, A.J. and Marcelino, M.J. (2014). A review of games designed to improve introductory computer programming competencies. *IEEE Frontiers in Education Conference (FIE) Proceedings*, Madrid, Spain, pages 1-7.

Wing, J.M. (2006). Computational thinking. *Communications of the ACM*, volume 49(3), pages 33–35.