PIDR - Rapport

Java Learning Machine

Encadré par Martin QUINSON et Gérald OSTER

Nancy-Université
Université Henri Poincaré

Sébastien TONI – Loïc VOURC'H

2010

Sommaire

1. Introduction

1.1. **Objectif du module**

Le *Projet Interdisciplinaire ou Découverte de la Recherche* est un projet de 6 mois servant d'introduction aux élèves de deuxième année à l'univers de la recherche. Dans ce module, contrairement aux projets standards, les élèves sont confrontés à un problème pour lequel il n'existe pas encore de solution explicite. Ils doivent donc explorer des pistes afin de trouver la bonne approche pour résoudre efficacement le problème.

1.2. **Contexte**

Java Learning Machine (JLM) est un outil pédagogique développé par Martin QUINSON et Gérald OSTER permettant aux étudiants d'apprendre les concepts de base de la programmation. Il est utilisé à l'ESIAL dans les modules PPP (Premier Pas en Programmation) et TOP (Techniques et Outils pour Programmer) enseignés en première année. La plate-forme est composée de nombreux exercices à difficulté progressive, couvrant l'ensemble des notions de bases de la programmation (introduction aux variables de données, aux instructions de boucle, aux déclarations et appels de méthodes, etc).

Les exercices sont construits de la manière suivante : une introduction avec un petit scénario, une leçon, et un objectif pour mettre en pratique ce qui a été appris. L'utilisateur acquiert donc pas à pas de nouvelles notions. La plateforme sauvegarde le code tapé par l'étudiant, lui permettant ainsi de revoir ce qu'il a fait, et potentiellement améliorer ce qu'il a fait.

Ainsi, *Java Learning Machine* introduit les élèves à des programmations de type procédural et impératif, ce qui constitue déjà une partie importante de la programmation. Cependant l'outil ne possède ni leçons ni exercices sur la programmation orientée objet.

1.3. **Mission**

Le premier objectif est d'améliorer la visualisation dans les exercices sur les algorithmes de tris. Le second objectif consiste à trouver l'approche pédagogique la plus appropriée à l'introduction à la programmation orientée objet, et de réaliser des leçons et exercices pour étendre la plate-forme. Les grandes notions de la programmation objet doivent être traitées (classes, objets, polymorphisme etc.). Le langage sélectionné est le même que celui qui est enseigné dans le module de POO (Programmation Orientée Objet) enseigné en première année à l'ESIAL, à savoir Java.

1.4. **Enjeux**

Les enjeux de ce projet sont multiples. Dans un premier temps les recherches menées doivent conduire à la réalisation de leçons et exercices qui seront utilisés par les élèves utilisant *Java Learning Machine.* Il en va donc de l'apprentissage d'élèves ingénieurs dans les langages objet. Dans un deuxième temps, les conclusions de la recherche doivent apporter une solution technique permettant l'implémentation d'autres exercices dans la suite du développement de la plate-forme. Ainsi mis en place, les nouveaux exercices pourront profiter aux futurs étudiants de l'ESIAL.

2. Analyse du problème

2.1. **Situation actuelle**

La plate-forme *Java Learning Machine* propose des exercices couvrant une grande partie des principes de la programmation non orientée objet. Celle-ci découpe les exercices selon différents thèmes pédagogiques. Chaque thème pédagogique contient une série d'exercices à difficulté progressive, portant sur un aspect précis de la programmation. Chacun des thèmes est composé d'un monde dans lequel évolue une entité solution proposée par l'élève. Cette entité est en réalité la forme sous laquelle le code de l'étudiant s'exécute.

Le monde est représenté graphiquement dans l'application dans une fenêtre située à droite de la zone de travail de l'étudiant. Cette représentation permet de visionner une démonstration de l'exercice et de représenter l'objectif à atteindre. Ainsi l'étudiant voit en direct l'effet de l'exécution de son code dans une fenêtre et l'objectif à atteindre dans l'autre.

De plus, certains thèmes pédagogiques sont abordés à l'aide d'un scénario mettant en scène les entités en les faisant évoluer dans un monde fictif. Ceci apporte un aspect ludique à l'étudiant et lui permet de mieux appréhender les concepts des leçons.

Le monde du thème sur les algorithmes de tri est actuellement représenté par le contenu des tableaux à trier. Chaque valeur est représentée par une barre de hauteur proportionnelle à la façon d'un histogramme. Ainsi, lorsque le tableau est trié les barres doivent apparaître dans un ordre croissant de taille. Ces dernières deviennent bleues lorsque l'objectif est atteint. Il serait intéressant d'intégrer une seconde vue, implémentant la nouvelle visualisation des algorithmes de tri conçue par le développeur Aldo Cortesi.

2.2. **Contenu à apporter**

Dans un premier temps, en guise de prise en main de l'environnement, il nous a été demandé de développer pour les exercices sur les algorithmes de tris une visualisation similaire à celle inventée par Aldo CORTESI. Tandis que la visualisation actuelle des exercices affiche l'état courant du tableau à trier, la méthode d'Aldo CORTESI propose de visualiser les opérations effectuées sur le tableau, depuis sont état initial jusqu'à l'état courant. Le système de visualisation d'Aldo CORTESI doit être implémenté. Ce dernier doit absolument s'intégrer à la plate-forme sans aucun effet de bord, il ne doit pas alourdir la représentation de l'exécution du code des étudiants. Il doit apporter une compréhension immédiate. Par ailleurs, ce dernier est un ajout, il doit donc être proposé comme une représentation supplémentaire et non pas remplacer la vue actuelle de l'exécution des algorithmes de tri.

Dans un deuxième temps, les concepts de la programmation orientée objet doivent être intégrés. L'implémentation de la programmation objet dans *JLM* doit se faire selon le protocole d'usage de la plate-forme. Elle doit apporter progressivement à l'élève les notions du langage objet (en l'occurrence Java). Pour ce faire, elle doit proposer des leçons d'introduction à chaque exercice. Ces dernières doivent balayer l'ensemble des notions nécessaires sans pour autant se plonger dans un niveau de détails trop poussé afin de ne pas rebuter l'étudiant. Ces leçons doivent donc être pertinentes et simples tout en étant complètes.

Le résultat doit permettre une utilisation de la plate-forme dans le cadre du module de Programmation Orienté Objet enseigné en première année à l'ESIAL. Ce qui apporterait une persistance dans le travail effectué par les élèves puisque ces derniers pourraient alors utiliser le système de sauvegarde de l'outil afin de reprendre leur travail où ils l'avaient laissé.

Ainsi, l'objectif du projet est d'introduire les utilisateurs de la plate-forme *Java Learning Machine* à la programmation orientée objet avec les concepts de *FUDC* et OOD en trouvant la meilleure approche possible. Une fois l'approche adoptée, il faut la concrétiser en créant des exercices et des leçons pour l'apprentissage des élèves. Enfin, une solution technique permettant d'implémenter les exercices dans le logiciel *Java Learning Machine* doit être apportée.

3. Organisation

3.1. **Outils**

Les outils utilisés pour ce projet sont tous informatisés. Parmi ceux-ci, certains sont utilisés à des fins communicatives, les autres concernent quant à eux la programmation des exercices sur la plate-forme.

3.1.1. **Communication**

La communication a joué un rôle crucial dans ce projet, aussi bien pour programmer que pour synchroniser le contenu des exercices ou encore le partager avec les encadrants.

Pour rédiger les nouveaux exercices sur la POO, nous avons utilisé l'application *Google Document.* Deux documents ont ainsi été mis en ligne. Un premier document faisant office de brouillon pour les nouvelles idées et les notes personnelles datées à chaque séance de travail. Un deuxième document, partagé avec les professeurs encadrant, contenant une mise au propre des derniers exercices rédigés.

Afin de récupérer la version originale de *Java Learning machine* située sur la plate-forme de travail collaboratif Assembla, nous avons utilisé *Git*. *Git* est un logiciel de gestion de versions décentralisé.

Après avoir récupéré la version originale de *JLM*, nous ne pouvions pas travailler directement sur le même dépôt *Assembla*, ce qui aurait remplacé la version originale par la nôtre. Afin de conserver le même schéma de travail et dans le but de synchroniser nos travaux, nous avons utilisé *Xp-Dev. Xp-Dev* est très similaire à *Assembla*. En effet, *Xp-Dev* une plate-forme collaborative privée. Cet outil permet de synchroniser des systèmes de fichiers distincts et donc de fusionner les travaux de programmation qu'ils soient effectués sur la même machine ou non. Utilisé seul, *Xp-Dev* est inutile, il est nécessaire de l'associer à une technologie de gestion de versions. Pour cette dernière nous avons choisis d'utiliser *Subversion.*

3.1.2. **Programmation**

Pour la programmation, il fallait utiliser un outil compatible avec *Subversion* et supportant le langage de programmation Java dans lequel est écrit JLM. Cet outil devait également fournir la possibilité de produire des patchs avec Subversion. Nous avons donc naturellement choisis d'utiliser l'environnement de développement *Eclipse*.

3.2. **Répartition des tâches, planning**

Les tâches à effectuer ont été équitablement partagées au sein du binôme. Le travail a principalement été réalisé durant des séances de travail se déroulant les mercredis et les jeudis après-midi. De plus, le projet étant totalement synchronisé par le biais des outils de communication mis en place, nous avons pu travailler parfois séparément à des heures réparties de façon irrégulière pendant le week-end.

Chaque semaine, un mail de rapport a été envoyé aux encadrant afin de faire le point sur l'évolution du projet et éventuellement poser des questions dans le but de résoudre d'éventuelles difficultés.

Des réunions ont également eu lieu au LORIA avec les encadrant de projet afin de faire la démonstration du travail effectué et de faire le point sur la direction à poursuivre.

4. Résolution

4.1. **Prise en main de l'environnement technique**

Nous avons d'abord développé une visualisation pour les algorithmes de tris calquée sur la méthode proposée par Aldo Cortesi. La vue est composée de plusieurs lignes. En abscisse, on a les opérations effectuées, en ordonnée, la position dans le tableau. La couleur de la ligne correspond à sa valeur (par exemple, pour un tableau de 10 valeurs, 0 correspond au noir et 9 au blanc). On peut ainsi observer les différentes positions dans le tableau que prendra une valeur avant d'atteindre sa place finale lors du tri. Une fois trié, la partie droite de la visualisation doit correspondra à un dégradé de couleurs.

Dans un premier temps, nous avions adopté une solution simple mémorisant l'état du tableau à trier à chaque opération. Malheureusement cette solution s'est révélée trop gourmande en mémoire. Nous avons donc opté pour une solution légèrement plus coûteuse en calculs mais permettant de limiter l'utilisation de la mémoire. Désormais, nous stockons l'état initial du tableau, ainsi que la liste des opérations effectuées par l'étudiant sur le tableau (insertion, inversion, etc.). Lors de l'affichage, nous générons dynamiquement la visualisation en appliquant successivement les opérations sur le tableau initial.

4.2. **Approche pédagogique**

Le paradigme de programmation objet a vu le jour en 1970 et le langage Java en 1995, ce qui fait maintenant une quarantaine d'années pour les langages objet en général et une quinzaine d'années pour Java. Malgré tout le temps écoulé depuis ces faits historiques dans le milieu de l'informatique, les méthodes d'approches d'enseignement de la Programmation Orientée Objet sont toujours aussi variées et indécises. Personne n'est actuellement en mesure de donner en toute objectivité la meilleure approche pédagogique pour enseigner ce concept. Il en existe de multiples.

De nombreux enseignants et chercheurs discutent des méthodes d'enseignement à utiliser pour transmettre aux mieux les grandes notions de la POO. Afin d'échanger les avis, évaluer et publier les résultats de leurs différentes expérimentation, le SIGCSE a mis au point une typologie. Le SIGCSE, pour *Special Interest Group on Computer Science Education* est un groupe international rassemblant des enseignants et des chercheurs qui

mettent en communs leurs différentes méthodologies d'enseignement concernant les sciences informatiques. La typologie créée par le SIGCSE découpe en trois parties l'enseignement de la programmation. Une première partie s'appelle FUDC (First User Defined Class) qui correspond à la première classe que l'utilisateur (dans notre cas l'élève) définit. Une deuxième partie s'appelle OOD (Object Oriented Design) qui définit la conception orientée objet. Enfin, une dernière partie se nomme CS (Control Structures), elle correspond entre autres aux instructions de type conditionnel et aux instructions de boucle. La partie OOD est décomposable en trois sous partie. En effet la conception orientée objet est composée entre de trois concepts majeurs que sont l'*héritage,* le *polymorphisme* et la *composition* et *collaboration* entre les objets. Ainsi, on peut écrire OOD-I, OOD-P, OOD-CC pour désigner respectivement les notions d'implémentation (héritage, interface), de polymorphisme et de collaboration et composition objet.

Java Learning Machine propose déjà d'acquérir des bases solides concernant la partie CS. En revanche, la plate-forme ne propose pas de FUDC, ni de OOD puisque cette dernière n'est pas encore programmée pour, d'où l'objectif que nous nous sommes vu fixé.

4.2.1.**Control Structures**

Nous avons donc décidé d'établir une approche pédagogique en commençant par aborder les notions de déclaration, instanciation d'objets et l'appel de méthode sur ces derniers. Ainsi nous commençons l'enseignement de la POO en traitant des notions de type CS (Control Structures), qui est une notion plus que développée dans les exercices déjà implémentés dans le JLM. L'élève n'est donc pas dérouté puisqu'il a déjà effectué de nombreux exercices de ce type.

4.2.2.**First User Defined Class**

Après avoir introduit ces premières notions de type CS, les élèves savent comment instancier et utiliser les objets. Ceci nous permet de converger progressivement vers l'introduction au premier exercice de type FUDC (First User Defined Class). Désormais l'étudiant doit apprendre à créer se propres classes d'objets avant des les utiliser. L'exercice consistera à faire écrire à l'élève sa première classe, la classe sera composée de méthodes et attributs simples pour commencer.

4.2.3.**Object-Oriented Design**

L'étudiant sachant à présent écrire le code de ses propres classes et donc concevoir ses propres objets, il est alors judicieux d'élargir ses connaissances quant à la façon de le faire. Nous décidons donc de continuer l'enseignement en abordant la

notion d'OOD-I. L'OOD-I traitant de l'héritage on peut désormais apprendre à l'élève tous les aspects liés au typage (statique et dynamique), à la redéfinition, à l'abstraction. Ceci permet d'aborder implicitement quelques premières notions de polymorphisme sans que ce dernier ne soit utilisé directement. Ici il s'agira entre autre d'écrire le code d'une classe héritant d'une autre. On peut aussi demander à l'élève d'implémenter des interfaces pour expliquer la différence entre les deux types d'implémentation (interface et classe mère).

Il conviendra bien évidemment de ne pas aborder toutes ces notions trop vite, mais au cours de plusieurs exercices et avec une difficulté très progressive. On peut facilement imaginer un voire deux exercices minimum pour certaines notions qui peuvent se révéler complexe.

Enfin, et pour finir concernant l'OOD, les notions qui selon nous seraient idéales à ce niveau de l'apprentissage seraient celles qui s'inscrivent dans l'OOD-CC. La collaboration, la composition ainsi que la délégation entre les objets sont des aspects essentiels. Sans ces derniers, les objets n'ont aucune interaction et perdent leur pouvoir de modélisation proche de la réalité qui fait leur charme. Il est donc indispensable de bien enseigner ces concepts à l'élève afin qu'il puisse coder des applications dans lesquelles il y a plusieurs entités interagissant. Or, à ce stade de l'apprentissage, l'étudiant possède tous les outils nécessaires à la compréhension de ces notions de collaboration, composition et délégation.

4.3. **Implémentation de la POO**

4.3.1. **Choix d'un univers**

Dans un premier temps, nous avions décidé de créer un univers spécifique à l'enseignement des langages objets. Il s'agissait d'un univers de type *Tower Defense*, cet univers présentait un avantage considérable du point de vue ludique et au niveau des certaines notions. En effet, le *Tower Defense* intègre très bien les notions de composition, collaboration et délégation. On peut imaginer assez facilement comment ces concepts viennent se greffer à l'histoire de cet univers. Par exemple, une tour voulant ouvrir le feu sur un monstre, délègue à ses canons qui la composent d'ouvrir le feu. Cependant, cet univers s'adaptait mal aux notions d'héritage, il était donc bien évident qu'il fallait le revoir.

Après avoir exploré différentes pistes, nous avons finalement décidé de fusionner le *Tower Defense* avec l'univers initial de *BuggleQuest*. Il est en effet possible d'ajouter de nombreux éléments à cet environnement. Le fait de conserver cet univers permet de garder une certaine cohérence, car les exercices viennent s'inscrire dans la continuité de ce que l'étudiant a déjà appris. En effet, *BuggleQuest* est un univers utilisé entre autre dans le thème pédagogique « Premiers pas » de JLM

qui est le tout premier thème pédagogique abordé sur la plate-forme. L'univers *BuggleQuest* met en scène des créatures ludiques que l'on nomme des *buggles.* Les buggles doivent accomplir différentes tâches telles que ramasser des objets, dessiner des motifs, sortir d'un labyrinthe etc. L'étudiant produit alors le code nécessaire pour guider les buggles et les faire interagir. Cet univers apporte une touche concrète à l'enseignement des bases de la programmation en mettant en scène des acteurs fictifs ludiques permettant à l'élève de mieux percevoir les effets du code qu'il produit.

4.3.2. **Implémentation de l'univers choisi**

L'implémentation de l'univers des Buggles dans le thème pédagogique de la POO doit se faire selon l'approche pédagogique que nous avons construite dans la partie 4.2 (*Approche pédagogique*). Nous devons donc introduire dans le même ordre chronologique les différentes notions exposées en 4.2 en les illustrant à l'aide de l'univers des buggles. Nous avons établi des exercices respectant l'approche pédagogique. A chaque nouvelle notion est associé un ou plusieurs exercices. Ce qui donne une leçon sur la programmation objet recouvrant toutes les concepts de l'approche comme suit.

- **Exercice d'instanciation, déclaration d'objets et invocation de méthodes**

Dans ce premier exercice, l'étudiant déclare et instancie une buggle de la classe MiniBuggle qui a été créée à cet effet. Des méthodes doivent ensuite être invoquées sur l'objet MiniBuggle ainsi créé. Le MiniBuggle doit attraper et déplacer un *Baggle* (gâteau ressemblant à un donuts dans l'univers des Buggles) à un endroit précis du monde.

Cet exercice peut être découpé en plusieurs de plus petites tailles et dont les objectifs sont plus élémentaires si les élèves rencontrent des difficultés. On peut imaginer dans un premier temps de simplement demander à l'élève de créé une MiniBuggle et de la placer dans le monde à des coordonnées précises à l'aide de simple méthodes setters. Ensuite, on pourrait créer un deuxième exercice où cette fois l'étudiant invoquerait plus de méthodes sur la MiniBuggle afin de déplacer le Baggle.

Remarque : On peut remarquer que cet exercice ne fait que reprendre des consignes des exercices du thème pédagogique « Premiers pas », ce qui est normal puisque les exercices de ce thème travaillaient sur les notions de CS (Control Structures) que nous commençons par aborder dans notre leçon afin que l'étudiant se base sur ses précédents acquis. Ainsi, nous reprenons des situations de l'univers déjà utilisées dans « Premiers pas » mais en nous élevant à un niveau

d'implémentation supérieur puisque l'étudiant déclare explicitement ses propres objets et ne se contente plus seulement d'invoquer des méthodes.

- **Exercice de FUDC 1**

Cet exercice est crucial, c'est le premier contact de l'étudiant avec le code d'une classe d'objet. L'élève doit ici écrire le code de sa propre classe que l'on nommera Bug. La classe Bug est en fait une espèce de Buggle. Ainsi, l'étudiant créé sa propre Buggle en implémentant un certain nombre de méthodes qui lui sont imposées. Ces méthodes doivent rester simples, il ne faut pas oublier que ce sont les premières méthodes de classes que l'élève écrit. Les attributs de la classe restent très basiques, on donnera à l'objet Bug quatre attributs entiers représentant sa position dans le monde (x, y) ainsi que la taille du monde (worldX, worldY). Les méthodes à écrire sont alors de simples méthodes de modification et d'accès (*getters* and *setters*).

Remarque : Cet exercice est l'occasion idéale d'introduire certaines conventions d'écritures inhérentes à Java à propos des noms des attributs et des méthodes.

- **Exercice de FUDC 2**

Dans cet exercice, il faut compléter le code de la classe Bug écrite précédemment. Il s'agit là d'ajouter des méthodes permettant au Bug de se déplacer. Il conviendra alors d'écrire quatre méthodes de déplacement relatif (moveUp(), moveLeft(), etc.). Suite à cet exercice on peut en insérer un autre demandant à l'étudiant d'instancier plusieurs objets de type Bug et de les déplacer à un endroit particulier du monde. Cet exercice mettrait en évidences l'utilisation de plusieurs objets simultanément.

- **Premier exercice d'OOD-I**

S'inscrivant dans une parfaite continuité avec l'exercice précédent, celui-ci demande à l'élève de redonner à sa classe Bug les méthodes déplacement relatif, mais cette fois, à moindre effort. En effet, on demande ici à ce que la classe Bug hérite de la classe SimpleBuggle. Cet énoncé introduit l'héritage à l'élève qui peut alors en comprendre les bases et illustre très bien l'intérêt d'une telle opération puisque cette dernière va lui procurer une définition toute faite des méthodes qu'il a durement implémentées à l'exercice précédent. De la sorte, l'apprenti comprend facilement le principe de factorisation qui se cache derrière l'héritage.

- **Approfondissement de l'OOD-I**

Le premier exercice d'OOD-I met en évidence la possibilité de factoriser du code propre à plusieurs entités ayant un point commun (ici les SimpleBuggles et les Bugs écrit par l'élève). Il est donc temps de révéler à l'étudiant d'autres atouts qu'apportent l'héritage.

L'élève doit agrandir les fonctionnalités des Bugs par rapport aux SimpleBuggles en leur ajoutant de nouveaux attributs et nouvelles méthodes. Les Bugs auront désormais des points de vie qui devront décroître lorsque ces derniers se déplacent sur le monde. Afin de regagner leurs points de vie, ils devront manger des Baggle. Si les points de vie d'un Bug atteignent zéro, le Bug meurt.

Remarque : En partant de cette nouvelle implémentation, on peut imaginer de nombreux exercices. On peut envisager par exemple de reprendre une série d'exercices du thème pédagogique « Labyrinthes » dans lequel les Buggles devaient trouver la sortie des labyrinthes. Avec leurs nouveaux attributs, un nouvel enjeu apparait alors, les Buggles doivent sortir vivant, ce qui amène le facteur efficacité algorithmique au premier plan. En effet, si l'algorithme de l'étudiant possède une mauvaise complexité le rendant inefficace la Buggle mourra avant d'avoir trouvé la sortie du labyrinthe car elle aura effectué trop de déplacements superflus.

On peut également contrôler le bon comportement de l'algorithme de l'étudiant en imposant des points de passages implicitement dans le labyrinthe en mettant des Baggles à des points clés stratégiques. La Buggle devrait alors manger le baggle en passant dessus s'il veut regagner ses points de vie.

- **Exercices d'introduction de l'OOD-CC (collaboration)**

Après tous ces exercices, il est temps que l'étudiant conçoive des classes contenant de la composition et que ses classes effectuent de la collaboration. C'est ce que nous proposons de faire dans cette partie de l'apprentissage en demandant l'écriture de nouvelles classes, codant des objets qui vont progressivement interagir avec les Bugs. C'est d'ailleurs à ce stade, que nous incluons les travaux effectués sur le thème du *Tower Defense*. En effet, nous demandons désormais à l'élève de créer des objets de terrain tels que la parcelle de sable ralentissant le Bug ou encore le téléporteur qui comme son nom l'indique téléportera le Bug d'un point du monde à un autre. Ainsi, l'étudiant va devoir écrire des méthodes faisant interagir des objets de natures différentes. Il s'agit donc dans un premier temps de collaboration plus que de composition.

- **Exercices d'OOD-CC (composition et délégation)**

Tout en continuant à faire évoluer l'univers des Buggles vers un univers hybrides de Bugs et de *Tower Defense*, il est maintenant question d'écrire des classes de tours et de passer du côté obscure en attaquant les Bugs avec ces tours. Les tours sont composées de canons qui sont eux aussi des objets à concevoir. C'est ici que sont donc évoquées les premières notions de composition d'objet. Une tour va ordonner à ses canons d'ouvrir le feu en direction des Bugs qui passeront à proximité.

Ainsi l'élève ne se préoccupe plus du sort des Bugs mais va désormais empêcher ces créatures d'atteindre un certain point du monde à l'aide des tours et des parcelles de terrain qu'il a créé lors de l'exercice précédent, qui sont maintenant plus de l'ordre du piège. On peut donc facilement imaginer des exercices dans lesquels les Bugs veulent piller un ou plusieurs baggles que l'étudiant protège à l'aide de ses tours et pièges.

- **Exercices d'approfondissement et d'OOD-P**

L'idée à ce stade de l'apprentissage est de créer toute une série de tours aux caractéristiques particulières pour faire face à l'évolution des Bugs. Il s'agit donc d'Exploiter le système d'héritage au maximum. On peut ainsi découper les tours en plusieurs familles telles que les tours faisant des dégâts selon une trajectoire, d'autres visant qu'un point précis du monde ou encore certaines faisant des dégâts sur des zones. La dualité entre la conception scénaristique de l'univers et la conception objet des tours est alors assez immédiate. En effet, on voit très bien le découpage en différentes lignées d'héritage des classes qu'il est alors possible de demander à l'élève. C'est à ce moment précis que peut être abordé sereinement la notion de polymorphisme.

- **Exercices divers**

Une fois tous les exercices précédents effectués. L'élève possède des bases dans les notions FUDC, OOD et CS. On peut donc étendre à volonté l'univers des Bugs et du *Tower Defense* pour pousser l'interaction entre les différentes notions de la POO et exercer encore plus l'élève.

4.3.3. **Notions couvertes**

Etant donné que l'objectif de l'implémentation des langages objets est d'être autonome et indépendant des enseignements à l'ESIAL, nous devons nous assurer que toutes les notions enseignées en POO sont couvertes.

TD/TP	Notions couvertes	Equivalent dans JLM
TD1, TP1, TP2	FUDC	Création de la classe Bug
TD3	CS	Premiers pas
TD4	OOD-CC	Tours, terrains, créatures
TP5, TD5, TP6, TD8	OOD-I	Exercices avec les Bugs
TD6, TD8	OOD-P	Exercices avec les tours

Comme on peut le constater, chaque notion vue dans le module de POO a son équivalent dans JLM.

4.4. **Considérations techniques**

Afin de traiter les exercices d'instanciation, nous avons créé des classes permettant de récupérer les objets instanciées par l'élève. Dans l'exemple d'exercice, l'étudiant doit instancier un objet de type MiniBuggle puis appeler des méthodes sur cet objet. Les exercices héritent d'une entité particulière qui possède un attribut de type MiniBuggle. On utilisera par exemples les méthodes setX() et setY() pour déplacer de manière absolue la buggle. On fera ensuite des exercices dans lesquels l'étudiant utilise les méthodes forward(), turnLeft() et turnRight() afin de se déplacer et d'aller chercher un baggle.

La grosse difficulté dans l'extension aux langages objets réside dans le fait de pouvoir travailler avec le code que l'étudiant a tapé afin de corriger son travail. Aussi il fallait trouver une solution permettant de récupérer les classes écrites par l'élève afin de pouvoir faire des tests dessus. Pour cela, nous avons utilisé des Meta-mondes. Ces Meta-mondes sont déjà utilisé dans les exercices sur les tours de Hanoï. L'élève ne travaille plus sur des entités mais directement sur des mondes.

5. **Conclusion**

5.1. **Résultats actuels**

L'ajout du système de visualisation d'Aldo Cortesi rend le résultat de l'exécution des algorithmes de tri plus attrayant et permet de discerner avec beaucoup plus d'efficacité les différents types d'algorithmes uniquement à l'aide de la visualisation et de déceler de potentielles anomalies. De plus, la présence de la création d'Aldo Cortesi dans JLM lui permet d'être au nec plus ultra des algorithmes de tri.

La structure proposée permet d'ajouter un grand nombre d'exercices couvrant les différentes notions de la POO. Le nouveau module s'inscrit dans la continuité de *BuggleQuest* et permet donc de conserver une cohérence dans l'environnement. De même, l'approche incrémentale se basant sur les acquis de l'élève renforce la cohérence de l'enseignement des nouvelles notions.

Au fil de l'avancement du projet, nous avons réalisé qu'il n'était vraiment pas facile de prendre des décisions en termes de pédagogie. Il est essentiel de garder à l'esprit que l'enseignement doit rester ludique, et que l'étudiant doit avoir envie de lui-même d'avancer et d'apprendre. Il faut également faire en sorte d'introduire progressivement chaque notion et de rester globalement cohérent.

5.2. **Perspectives**

La structure actuelle utilise l'univers de *BuggleQuest*. On pourrait créer d'autres univers permettant d'illustrer les différents TDs du module de POO. Il serait également intéressant de développer une visualisation de l'état de la mémoire avec une représentation des objets sous forme de boîtes, comme ce qui est vu en cours et TD. L'étudiant pourrait ainsi visualiser ce qu'il a réalisé et mieux assimiler les notions vues en cours. Combiné aux exercices déjà présents, JLM deviendrait alors l'outil principal pour les TPs du module de POO. JLM pourrait même devenir l'outil utilisé pour les TPs notés. Renvoyant une correction instantanée et automatisant la notation.

On peut également ajouter un module permettant à l'enseignant de voir quels élèves sont en difficulté, en repérant par exemple les élèves ayant passé trop de temps sur un exercice sans le résoudre, ou ayant essayé un trop grand nombre de fois son code sans succès.

Glossaire

POO : *Programmation Orientée Objet*. Style de programmation informatique.

JLM : *Java Learning Machine*. La plateforme d'apprentissage des bases de la programmation.

SIGCSE : Special Interest Group on Computer Science Education. Groupe spécialisé dans l'enseignement de l'informatique pour professeurs et chercheurs.

CS : *Control Structures*. Il s'agit de la programmation.

OOD : *Objet Oriented Design*. Le fait de mettre en place une interaction entre les objets afin de résoudre un problème donné.

OOD-I : Sous partie de OOD consacrée à la notion d'héritage.

OOD-CC : Sous partie de OOD consacrée aux notions de collaboration et composition.

OOD-P : Sous partie de OOD consacrée à la notion de polymorphisme.

FUDC : *First User Defined Class*. Une des premières notions de l'apprentissage des langages objets.

Buggle : Il s'agit d'une entité manipulée dans les premiers exercices de JLM.

Baggle : C'est un objet que les buggles doivent ramasser dans JLM.

BuggleQuest : C'est l'univers utilisé pour les premiers exercices d'initiation.

Bibliographie

Jaime NINO, Frederick A. HOSCH, *Introduction to Programming and Object Oriented Design using Java*, Wiley, 2008.

Cay HORSTMANN, *Big Java*, Wiley, 2008.

Jürgen BÖRSTLER, Mark S HALL, Marie NORDSTRÖM, James H PATERSON, Kate SANDERS, Carsten SCHULTE, Lynda THOMAS.
"An Evaluation of Object Oriented Example Programs in Introductory Programming Textbooks", SIGCSE Bulletin Volume 41, Number 4, decembre 2009
p. 126-141