Rapport préliminaire de PIDR

Par Sébastien CHAMPAGNE et Gabriel LARROQUE
encadrés par Martin QUINSON et Gérald OSTER

19 mai 2011

Table des matières

Chapitre 1

Introduction

1.1 Objectifs du module PIDR

Le PIDR -*projet interdisciplinaire ou découverte de la recherche*- constitue une première approche de la recherche dans l'enseignement dispensé à l'ESIAL. Ce projet se distingue des autres car en plus de consolider nos compétences, il donne une vraie chance d'apporter une contribution à l'équipe de recherche qui propose de PIDR.

1.2 Contexte

Pour nous, ce projet a consisté à contribuer à l'amélioration d'une plateforme d'aide pour l'apprentissage de la programmation en JAVA, JLM -*JAVA Learning Machine*. Martin QUINSON et Gérald OSTER, les développeurs de l'application, ont supervisé ce PIDR.

Le principe de l'application est d'écrire un morceau de code pour interagir avec "un monde". Les mondes sont des environnements avec une représentation graphique et une histoire, ce qui ajoute une dimension ludique à l'apprentissage. Le monde qui nous intéresse le plus dans le cadre de ce PIDR s'appelle BuggleQuest. C'est une grille sur laquelle des entités, les buggles, se déplacent d'une case à l'autre.

Le logiciel compte déjà un grand nombre de leçon pour apprendre à programmer. Les exercices travaillent

- sur les structures de contrôle de base(boucle,condition,tableau,...)
- sur les tris
- sur la programmation récursive.

Notre sujet de PIDR s'inscrit dans la continuité du travail déjà réalisé sur l'application. Nous devons étudier les possibilités d'utilisation de la JLM pour l'apprentissage de la programmation orientée objet -POO. Toutefois nous pouvons nous appuyer sur les travaux du groupe de PIDR de l'année dernière de Loïc VOURCH et Sébastien TONI. Il ont travaillé sur le même sujet et ont analysé les techniques actuelles d'enseignement de la POO. Nous nous sommes aussi inspirés du scénario d'apprentissage qu'ils avaient commencé à développer même s'il était resté à l'état de projet dans leur rapport.

À partir de cette base, nous avons donc établi notre propre suite d'exercices et les avons déployés sur JLM.

Chapitre 2

Protocole

2.1 Affichage des constantes du buggle

La première étape du projet a été de se plonger dans le code de la JLM développé par Martin QUINSON et Gérald OSTER. Pour s'approprier rapidement ce code nous avons commencé par implémenter une fonctionnalité pour l'application. Cette fonctionnalité consiste en l'affichage de différentes constantes dans le monde BuggleQuest de la JLM, comme, par exemple, les résultats des fonctions *getX()* et *getY()*, telles que l'élève peut les obtenir dans son code.

Cette première approche a été purement technique pour ainsi mieux comprendre l'architecture du logiciel et les outils déjà à notre disposition dans le logiciel.

2.2 Une leçon POO

Ensuite nous avons préparé la dernière phase de notre protocole en mettant en place une leçon POO initialement vide et prête à recevoir les exercices que nous allions développer. Nous nous sommes servis de ce que nous avait appris la première phase du protocole pour insérer convenablement cette leçon dans l'architecture du logiciel.

2.3 Les exercices

Pour terminer, nous nous sommes servis du travail du groupe de PIDR de l'année dernière(Loïc VOURCH et Sébastien TONI) pour mettre en place une stratégie pédagogique pour enseigner la POO par une suite d'exercices qui puisse s'ajouter à la leçon POO qu'on avait précédemment créé dans l'application. Il s'agit de la phase la plus importante du PIDR. Ceci demande de mettre le problème en perspective suivant les dimensions

- didactiques
- ludiques
- de vraisemblance dans la mythologie du monde de JLM choisi pour les exercices (en l'occurrence, nous avons choisi le monde BuggleQuest).

Chapitre 3

Résultat de notre travail

3.1 Affichage des constantes du buggle

3.1.1 Intérêt attendu

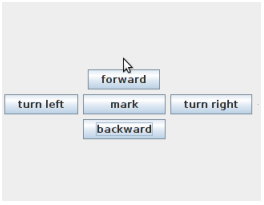
L'idée de cette fonctionnalité vient d'une constatation très simple. Tous les débutants en programmation n'ont qu'une façon de déboguer leur code : utiliser des *"System.out.println(...)"* un peu partout pour afficher l'évolution des constantes de leur programme. D'où notre idée d'afficher en permanence les constantes les plus triviales du monde, obtenues par les getters sur le buggle. En outre, l'autre utilité de cette affichage est de donner plus rapidement à l'élève la connaissance des méthodes disponibles sur le buggle pour récupérer des informations sur leur environnement.

En définitive, cette fonctionnalité permet, d'une part, une prise en main plus rapide par l'élève de l'interface offerte par le buggle, et d'autre part, elle offre une aide pour le débogage du code de l'élève.

3.1.2 Spécification et architecture

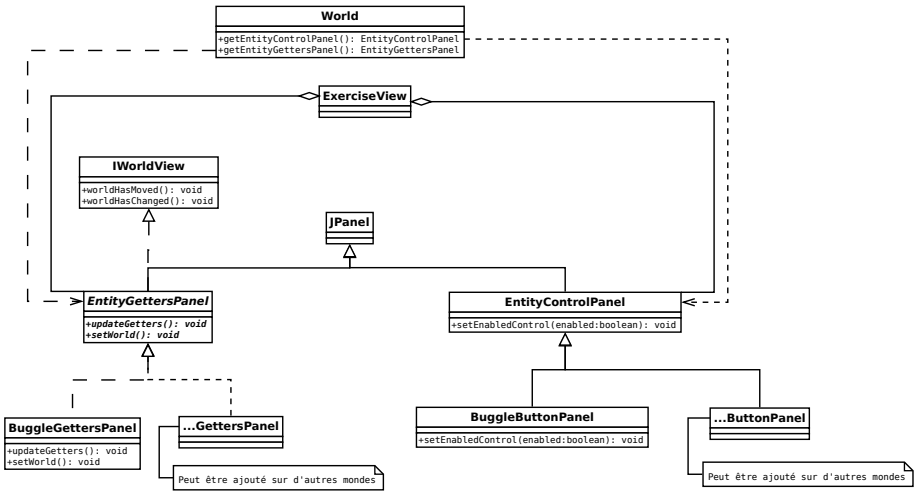
L'implémentation de cette fonctionnalité a été grandement facilité par l'existence d'une fonctionnalité similaire : le controlPane, qui est une collection de bouton pour faire bouger le buggle. Nous nous sommes donc inspiré de l'architecture de ce controlPane pour réaliser ce que nous avons appelé un gettersPane.

FIGURE 3.1 – BuggleButtonPanel



L'enjeu de l'architecture et de pouvoir dériver l'interface de la classe GettersPane pour pouvoir l'utiliser sur d'autres mondes que BuggleQuest. L'architecture du controlPane étant suffisamment agile, nous avons eu très peu de modification à y apporter.

FIGURE 3.2 – Diagramme de classe montrant les dépendances du GettersPanel



Après l'architecture, il fallait réfléchir au graphisme. Nous sommes resté simple en choisissant une ergonomie en tableau pour être le plus clair possible, les informations les plus importantes étant en haut du tableau.

Nous avons attaché de l'importance à garder le gettersPanel fonctionnel lorsqu'on change de leçon, d'exercice, de monde ou d'entité -les buggles- sélectionné. Il reste même à l'écoute de l'évolution du monde pendant l'exécution du code de l'élève.

En définitive, nous avons réussi parfaitement à atteindre nos objectifs pour cette fonctionnalité.

FIGURE 3.3 – BuggleGettersPanel

3.2 Une leçon POO

3.2.1 Mise en place de la leçon

Pour commencer, nous avons donc mis en place dans la JLM une leçon intitulé ***Premier Pas en POO***. Le premier choix à faire a été de décider dans quel monde inscrire la leçon. D'une part, le monde BuggleQuest est celui que nous connaissons le mieux puisque nous avons déjà travaillé dessus en implémentant le gettersPanel. Et d'autre part, le groupe de PIDR de l'an dernier a travaillé avec ce monde lui-aussi. Donc placer la leçon de POO dans le monde BuggleQuest nous a semblé être le choix le plus logique.

En outre, nous nous sommes simplement servis des autres leçons comme modèles pour modifier le fichier de configuration et charger cette leçon comme un plugin à l'instar de toutes les autres.

3.2.2 Organisation de la leçon

Pour établir une organisation des exercices de POO, nous nous sommes appuyés sur le rapport [1] du PIDR de l'an dernier sur le même sujet. Dans ce rapport, le cours de programmation orientée objet est découpé en trois parties :

- CS -*control structures*- qui concerne l'apprentissage des structures de programme -non spécifique aux langages orientés objet et déjà assuré par les autres exercices déjà implémentés dans la JLM. Nous avons donc décidé de se concentrer sur les 2 autres parties.
- FUDC -*First User Defined Class*- qui consiste à apprendre la structure d'une classe indépendemment des autres.
- OOD -*Object Oriented Design*- qui est l'étude des architectures de programme en langage orienté objet *i.e* les principes d'héritage, d'interface, de polymorphisme, de délégation, ...

Cette décomposition est extraite de la typologie mise au point par le SIGCSE -Special Interest Group on Computer Science Education- pour faciliter les échanges sur les stratégies d'enseignement de la POO.

De plus, le groupe de PIDR de l'an dernier a proposé un ordre chronologique pour les exercices à donner à l'élève. Nous nous en sommes inspirés pour découper notre leçon en 2 séries de 3 exercices de difficultés croissantes :

- 3 sur la structure d'une classe -FUDC- où il s'agit de former l'élève à écrire des classes avec des constructeurs des attributs et des méthodes. Ces trois exercices se terminent par l'implémentation du jeu de la vie.
- 3 sur l'héritage et le principe de délégation -OOD.

Ces exercices supposent que l'élève possède déjà les bases des structures de contrôle en programmation qu'il a acquises en suivant la leçon ***Premier Pas En Programmation*** qui est déjà dans la JLM. Ils seront détaillés dans les deux sections suivantes.

1. il est disponible à l'adresse `http://www.loria.fr/~quinson/Research/internships/2010_toni_vourch-rapport.pdf`

3.2.3 Exercices FUDC

Les exercices de la catégorie FUDC viennent donc naturellement en premier juste après les structures de contrôle -CS- vues dans ***Premier Pas En Programmation***. Ils suivent un ordre croissant de difficulté. La compétance qu'ils permettent de travailler est l'écriture d'une classe. Après ces trois exercices, l'élève est censé connaitre la structure d'une classe :

- Constructeurs
- Attributs
- Méthodes

FUDC1 : Ma première classe MyMiniBuggle

Ce premier exercice intitulé ***première classe***, est le moins ambitieux possible puisqu'il s'agit seulement de mettre en place un constructeur, deux attributs et une méthode très basique de type *"setter"*.

Attributs Les attributs sont des variables associées à un objet. Dans cet exercice, les deux attributs de la classe représentent les coordonnées de la case sur laquelle est le buggle. Les noms de ses attributs doivent forcement être posX et posY pour pouvoir être interprétés par l'environnement graphique.

Constructeur Le premier constructeur que l'élève devra écrire, initialisera, comme tout constructeur, les attributs de la classe. Nous prendrons 1 comme valeur initiale. Cette initialisation intervient automatiquement après l'instanciation d'un nouvel objet.

Méthodes Les méthodes de classe sont les fonctionnalités disponibles sur l'objet. La méthode setPos que l'élève doit écrire, est un *"setter"* qui change la position d'un buggle. C'est un type de méthode très classique en POO.

Instanciation En outre, nous demandons à l'élève d'écrire lui-même le code de la fonction *"main"* qui sera exécutée, pour lui rappeler les concepts de CS que nous avons supposé acquis. On peut regretter que nous n'ayons pas pu résoudre un problème avec le nom de la classe à compléter qui doit être changée par l'élève comme c'est indiqué dans l'énoncé de l'exercice.

FUDC2 : Une méthode plus compliquée, clone()

L'exercice suivant, ***multiclonage***, reprend la classe précédente et demande à l'élève d'ajouter une méthode qui renvoie un objet du même type que la classe qui définie la méthode. L'élève doit arriver à prendre un peu de recul sur la notion d'objet pour passer cet exercice. En effet, la duplication d'objet est un concept essentiel en POO. Le piège serait de penser que, à l'instar des types primitifs -int, boolean,char,...-, on peut dupliquer un objet *"a"* avec l'instruction *"b=a"*. Mais lorsque *"a"* est un objet, *"b"* représente alors le même objet i.e toute modification de *"a"* modifie aussi *"b"*. Une solution est de définir la méthode clonage sur l'objet, comme c'est proposé dans l'énoncé de cet exercice.

FIGURE 3.4 – Code de la méthode clonage

```
MyMiniBuggle clonage(){
        MyMiniBuggle mt = new MyMiniBuggle();
        mt.posX=posX;
        mt.posY=posY;
        return mt;
}
```

FUDC3 : Le jeu de la vie

Pour finir avec le FUDC, nous avons décidé de proposer un exercice, ***jeu de la vie***, plus difficile pour faire entrevoir les fonctionnalités avancées des concepts objets et comment les utiliser pour résoudre des problèmes plus complexes. Le jeu de la vie est un petit défi pour l'élève. Les règles sont simples et donc pas très longue à coder mais c'est un programme entier très classique. Ce jeu nous a donc semblé parfait comme ouverture pour encourager l'élève à avancer de lui-même dans l'étude du concept d'objet en programmation.

Principe Le jeu se déroule sur une grille à deux dimensions de 20 par 20, dont les cases — qu'on appelle des « cellules » peuvent prendre deux états distincts :

- « vivantes »
- ou « mortes ».

Règles À chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante :

- Si une cellule a exactement trois voisines vivantes, elle est vivante à l'étape suivante
- Si une cellule a exactement deux voisines vivantes, elle reste dans son état actuel à l'étape suivante
- Si une cellule a strictement moins de deux ou strictement plus de trois voisines vivantes, elle est morte à l'étape suivante

Une structure de code est disponible pour guider l'élève au début. Cela permet de ne pas le décourager trop vite.

3.2.4 Exercice OOD

Les classes sont les briques d'un programme en POO. Et après les exercices FUDC, l'élève doit, à peu près, savoir les fabriquer et les manipuler. À ce stade, il lui reste à apprendre comment assembler ces briques entre elles ; c'est OOD. Nous nous sommes concentrés sur trois grandes façons de faire interagir des classes entre elles :

- l'utilisation d'interface
- l'héritage
- la délégation

Nous avons aussi essayé d'insérer une progression par ordre de difficulté croissante comme pour FUDC.

FIGURE 3.5 – Interface DNABuggle

```
public interface DNABuggle{
        /**
         * Setter
         * @param x Position X
         * @param y Position Y
         */
        public void setPos(int x,int y);
                /**
                                *Getter
                        * @return int PosX
                 */
        public int getPosX();
                /**
                        *Getter
                 * @return int PosY
                 */
        public int getPosY();
}
```

OOD1 : Une interface DNABuggle

Cette exercice, ***ADN des buggles***, cherche à familiariser l'élève avec la notion d'interface. Dans un objectif pédagogique, nous avons comparé l'interface avec l'ADN du buggle qui comme chez l'être humain représente l'ensemble des fonctions primordiales de l'entité.

En POO, l'interface est un contrat qui engage le développeur à implémenter les méthodes dont le prototype est dans l'interface. La première interface à implémenter pour l'élève ne compte que des méthodes basiques :

- 1 *"setter"*
- 2 *"getters"*

OOD2 : L'héritage de MyBigBuggle

Le concept d'héritage est introduit dans le monde des buggles par le principe de l'évolution. ***Évolution des buggle*** va donc obliger l'élève à expérimenter la factorisation de code que peut apporter le mécanisme de l'héritage en programmation orientée objet.

Le principe de l'héritage est que la classe fille -MyBigBuggle- contient déjà tout le code de la classe mère -MyMiniBuggle- de façon implicite. On obtient ici un gain de temps en ne réécrivant pas tout le code soit-même à l'intérieur de la classe fille. De cette façon, l'élève peut se concentrer, immédiatement, sur la méthode *"mad"* à implémenter.

La règle de *"mad"* est simple :

- Si la position en x est pair, on incrémente de 1 la position en x
- Si la position en y est pair, on décrémente de 1 la position en y
- Si la position en x est impair, on décrémente de 1 la position en x
- Si la position en y est impair, on incrémente de 1 la position en y

L'élève a donc une occasion de revoir la structure de contrôle -CS- des conditions, en plus d'expérimenter le potentiel de l'abstraction de l'héritage en POO.

OOD3 : Délégation sur une collection de Buggle

Le principe de délégation abordé dans cette exercice, ***à vos ordre***, est plus compliqué que les deux notions OOD précédentes. L'élève va devoir abstraire le problème pour comprendre qu'il doit seulement donner l'ordre à un buggle sans se soucier de sa classe (Commander ou Performer).

Pour appliquer la délégation, il faut une collection d'objets -l'ArrayList BugglesPerformers- avec une interface commune -DNABuggle- sur lesquels déléguer une action -la méthode go(). L'implémentation concrète de la méthode go n'a pas d'importance pour la classe qui invoque la méthode. En d'autres termes, la délégation sert à réduire les dépendances entre les objets.

Dans cet exercice, il y a donc deux implémentations possibles pour la méthode go() :

- dans PerformerBuggle, le buggle écrit son message : *"I am a performer"*
- dans CommanderBuggle, le buggle écrit son message : *"I am a commander"* et invoque sur chacun de ses *"BugglesPerformers"* la méthode go().

En définitive, la leçon s'achève sur l'une des questions les plus importantes en programmation orientée objet : *l'architecture*.

Chapitre 4

Discussion

4.1 Le GettersPanel

Le bilan de cette fonctionnalité est très positive. Elle a bien rempli son rôle en nous aidant à nous approprier le code de la JLM. De plus, son fonctionnement dans le monde BuggleQuest est satisfaisant. Toutefois, ce gettersPanel est incompatible avec la leçon de POO que nous venons d'ajouter. La JLM n'a pas été conçu initialement pour exécuter du code sous forme de classe. Nous avons dû utiliser des ruses pour attacher la classe de buggle conçue et manipulée par l'élève à la vue qui apparait à l'écran. Et malheureusement, ces ruses court-circuitent le gettersPanel.

À part donc cette petite ombre au tableau, nous sommes parvenus à atteindre nos objectifs avec cette fonctionnalité. L'affichage se met à jour correctement lorsque l'élève change de leçon, d'exercice, de monde, ou de buggle sélectionné. Et comme il reste aussi actif pendant l'exécution du code, l'élève obtient des renseignements supplémentaires pour déboguer son programme.

4.2 Les exercices

Les exercices que nous proposons couvrent la base de la programmation orientée objet. Cependant, si nous voulions aller jusqu'à enseigner les concepts de design en POO, cela semblerait irréalisable sur JLM en l'état actuelle de l'outil puisque celui-ci ne permet de tester un code que sur son résultat et non sur son architecture. Même pour programmer de simple classe nous avons dû utiliser des ruses pour les attacher au monde en restant le plus simple pour l'élève. Il reste dans les exercices des élément gênant pour l'élève comme les appels un peu mystique à une fonction display mais il nous a semblé très dur de faire beaucoup mieux.

Toutefois nous sommes parvenus à offrir aux élèves un vue globale des notions de POO avec 3 exercices sur FUDC et 3 autres sur OOD.

4.3 Bilan sur l'exercice du PIDR

Le PIDR nous aura initié à la réalisation d'un projet de recherche. Le déroulement nous a semblé très proche des autres projets qu'on a fait à l'ESIAL. Seulement dans ce projet nous avons dû faire preuve de beaucoup plus d'initiative personnelle puisque personne ne connait la réponse au sujet.

Annexe

Glossaire

POO Acronyme pour *Programmation Orientée Objet*. C'est un paradigme de langage de programmation à l'instar des langages impératifs, fonctionnels,...

JLM Acronyme pour *Java Learning Machine*. Plateforme d'apprentissage pour les langages de programmation et en particulier tournée vers le JAVA.

SIGCSE *Special Interest Group on Computer Science Education*. Groupe de réflexion dans le domaine de la pédagogie en informatique.

CS *Control Structures*. Regroupe les compétences en informatique sur l'utilisation des structures de programmation classique comme les boucles et les structures conditionnelles. Première étape de l'apprentissage de la POO.

FUDC *First User Defined Class*. Il s'agit des notions sur la structure et l'utilisation -instanciation de nouveau objet, invocation des méthodes- d'une classe indépendamment des autres. Deuxième étape de l'apprentissage de la POO.

OOD *Objet Oriented Design*. Partie du paradigme de POO sur l'utilisation des relations possibles entre les classes -Héritage, collaboration,... Troisième étape de l'apprentissage de la POO.

BuggleQuest Un des monde de JLM. C'est une grille où vivent des créatures originales, les buggles. Les buggles se déplacent d'une case à une autre de la grille.

Buggle Entité vivant dans le mond BuggleQuest de JLM.

Bibliographie

PIDR-Rapport, Java Learning Machine, par Sébastien TONI et Loïc VOURC'H.http://www.loria.fr/~quinson/Research/internships/2010_toni_vourch-rapport.pdf