Baba is You

Développement et architecture du projet

RAMAROSON Johan
Groupe 3

GUYOLLOT Alan
Groupe 3

Du 28 Octobre au 16 Janvier 2021

Chargé de TD : Monsieur Forax
Matière : Java Programmation Orienté Objet

Etablissement / Formation :
Université Gustave Eiffel – Licence 3 Informatique Semestre 5

I/ Introduction

Le projet réalisé est le début d'un « remake » du vrai jeu « Baba is you » disponible sur **Steam**. Les règles y sont spécifiés dans le fichier ***user.pdf***. Ici nous parlerons de la partie programmation donc de la conception du projet.
On y spécifiera nos idées et nos choix, ce qui a été retenus et changés depuis la soutenance bêta, les difficultés rencontrés, les structures utilisés ainsi que l'architecture finale du projet. Le projet a été réalisé entièrement en Java en utilisant la librairie ***zen5.jar*** pour la partie graphique et interaction avec l'utilisateur.

II/ Architecture du projet

Division du projet en packages

Le projet est divisé en plusieurs packages présentés ainsi :

- ***Assets*** : Contient les dossiers images qui contient toutes les images utiles à la partie graphique du jeu, il contient également le dossier contenant les différentes énigmes du jeu. Ces fichiers seront lus par le programme selon une architecture bien spécifique. Il contient également une classe « PathAssets » gérant les chemins menant aux différents assets du jeu.

- ***Data*** : Contient tous les objets « matériels» utiles au jeu, on y retrouvera la gestion des blocs, des textes, ainsi que les énumérations de tous les matériels du jeu comme les **Nouns**, les **Operators** et les **Properties**.

- ***Deplacement*** : Contient tous les fichiers gérant la gestion des mouvements et des déplacements d'un personnages, la direction que prend un personnage lorsqu'il se déplace ainsi que les coordonnées d'un objet à l'écran.

- ***Graphics*** : Contient le fichier « **Renderer.java** » gérant la partie graphique du programme.

- ***Management*** : Ce package contient toutes les classes utiles au management du jeu, notamment a la gestion du plateau de jeu (***Board.java***), des règles présentent selon les textes en jeu (***Rules.java***), de la création du plateau (***MapGenerator.java***), ou encore le lancement de l'application (***Run.java***).

- ***Main*** : Contient la classe ***main.java*** qui contient la méthode *main* lançant l'exécution du programme.

Conception générale du jeu

Nous parlerons ici du fonctionnement du jeu en général et de nos structures de données utilisés, l'idée est de **dissocier les blocs des textes**, car les textes auront pour but de gérer les règles du jeu, les blocs de les appliquer.

Nous avons donc créé deux classes :
Bloc.java et ***Text.java***, ces deux blocs sont des éléments qui peuvent être représenté en jeu. Nous avons donc créé une classe abstraite reliant les deux et pouvant ainsi les dissocier et les utiliser différemment : ***Element.java.***

Nous utiliserons l'héritage donc, ***Bloc.java*** et ***Text.java*** hériteront de ***Element.java.***
Les matériaux sont des **Noun**, des **Operators** ou des **Properties.** Pour lister et dissocier chaque matériau, nous avons créé des énumérations : ***Noun.java***, ***Operator.java*** et ***Property.java***.

Ces 3 énumérations implémenteront l'interface **Material** qui permettra la dissociation des trois matériaux. Un matériau sera le champ d'un bloc ou d'un text. Cette implémentation permettra de dissocier si l'élément est un **bloc** ou un **text** et quel **matériau** il représente. Notre code permettra ainsi de faire fonctionner le programme différemment si on manipule un **text**, un **bloc** et selon le type de matériau.

Ces blocs seront donc représenter en jeu dans une certaine position à l'écran dépendant de leurs liste de coordonnées, c'est la classe ***Renderer.java*** qui s'occupera d'afficher et d'effacer les images à l'écran, via la méthode « drawElementByCoordinate » et « clearElementByCoordinate » qui affiche ou effacera l'image selon la coordonnée passé en argument, ainsi que la taille et la largeur de l'image.

Structures de données utilisées

Chaque Element en jeu possède une liste de position (définit par un ***hashMap<Coordinate, Integer>***) ainsi créée grâce a la classe ***Coordinate.java***, qui possède en champs **deux int : x et y**, désignant la position de l'élément en jeu.
Nous avons choisi d'utiliser une **HashMap** ici afin de contrôler la possibilité que deux coordonnées du même élément puissent être en conflit.

Les éléments auront une **hashSet** de propriétés, ceux ci permettront la définition des règles établis par le niveau en cours. La classe **Rules.java** permettra de définir les priorités et certaines règles concernant les propriétés, grâce au **hashSet** notamment nous pouvons également **contrôler les priorités des propriétés.**
Nous avons également décider d'utiliser une **hashSet** dans Element pour pouvoir controler les propriété qui seront ajouté grâce à la ligne de commande « --execute » (cf : user.pdf). Le principe est qu'elle ajoutera des propriétés dites « absolue » à l'élément donc qui ne pourront être suprimé autrement que de recommencer le niveau sans lancer l'execute.

Un plateau est définie grâce à la classe **Board.java.** Cette classe possédera un **hashSet** d'éléments, qui représentera tous les éléments du **board**. Grâce à ceci nous pourrons contrôler toutes les actions des éléments du jeu.

III/ Ajouts depuis la soutenance

Après la soutenance Bêta nous avons du ré-organisé notre projet, voici la liste des ajouts/suppressions :

Corrections / améliorations

- Ajout d'une classe ***Rules.java*** qui gère les règles de chaque élément
- Ajout d'une classe ***Renderer.java*** qui gère la partie affichage graphique des éléments du jeu
- Ajout d'une classe ***PathAssets.java*** qui gère le Path des assets du jeu (images, gifs, fichiers.txt pour les niveaux).
- Remplacement des classes ***Noun.java, Operator.java et Properties.java*** en énumérations appelés de la même manière.
- Ajout d'une interface ***Material*** qui sera implémenté par les classes ***Noun.java, Operator.java et Properties.java***
- Ajout des classes ***Bloc.java*** et ***Text.java*** qui permettront de dissocier les blocs jouables des textes
- Suppressions des **setters** et **getters « publics »** dans les classes, afin de contrôler notre code et utiliser le principe de la programmation objet via l'encapsulation
- Utilisation de **sets** au lieu **d'ArrayList** pour améliorer le coût des méthodes du code
- Découpage des méthodes pour ne plus en avoir avec + de 20 lignes
- Optimisation/réduction des boucles foreach, il n'existe plus de gros streams imcompréhensible, les méthodes deviennent plus claires à lire et à comprendre, nous avons donc réunis certains boucle dans une seule afin de ne pas parcourir plusieurs fois une structure de donnée.

Avancé du projet depuis la soutenance

Après la soutenance, nous avons donc ré-organisé le projet comme indiqué ci-dessus. Ensuite lorsque nous avons rattrapé le code, nous avons pu finir le projet en :

- commentant le programme
- créant du ***build.xml***
- créant le jar exécutable
- renommant les différents répertoires

Concernant notre propre niveau, nous avons ajouté les mots suivants : Virus (noun représenté par un petit virus vert à l'écran), Spread (propriété) et Vaccin(propriété).

Le Spread permettra à l'element la possédant dans leur set de propriété d'ajouter une coordonnée autour (east, nord, sud, ouest) de l'élément en question si et seulement si :
- la coordonnée ciblée n'est pas déjà pris par un autre element (l'élément spread peut donc être isolé).
- La coordonnée se trouve dans le plateau de jeu (il ne pourra pas s'échapper hors des limites du terrain) pour éviter les bugs et les ralentissements au bout de quelques touches.
La propagation de l'élément se fait à chaque fois que le joueur bouge d'une case (le virus se propage même si le joueur se cogne sur un element « Stop »).

IV – Conclusion

Ce projet nous a appris à utiliser nos connaissances acquise lors du semestre en java, sur un gros projet. Il nous aura également aidé à comprendre et manipuler plus en détail et en profondeur l'encapsulation ainsi que toute les spécificités de la programmation orientée objet.
La plus grosse difficulté aura été de reprendre le sujet en entier après la soutenance mais cette reprise aura été nécéssaire et utile, qui nous aura aidé non pas à faire fonctionner un programme mais à écrire du code propre et efficace.