ROUMEGOUS Adrien
SI Frédéric

Manuel développeur : « Baba is You »

Sommaire

Introduction

La base de ce projet est la conception du jeu « Baba Is You ». Ce jeu est une suite d'énigmes, séparées par niveaux, durant laquelle nous incarnons (pas toujours !) le personnage de Baba. Celui-ci doit parcourir un espace en deux dimensions afin d'atteindre son point d'arrivée. Petite spécificité du jeu, aucune règles (ou presque) n'est absolue ! Le joueur a accès à de différents mots qu'il peut manipuler à souhait afin de changer les fondements même du jeu. Ainsi, les murs peuvent ne plus être un obstacle, le drapeau qui signifie la victoire peut devenir votre plus grand danger, etc.

L'objectif ici est donc en tant qu'étudiants d'apprendre à manipuler la bibliothèque graphique zen5, ainsi que de reproduire un programme bien plus sérieux que les TPs vus jusqu'ici.

Architecture du programme

Le programme est constitué de seulement deux packages : fr.umlv.baba et… fr.umlv.baba.main qui contient la classe Main, chargée d'exécuter le programme.

Voici donc l'architecture du programme :

- Une classe **Board**, au coeur du programme. Elle contient le plateau de jeu contenant une représentation creuse de l 'ensemble des données, sous la forme d'une **HashMap** composée de tous les types d'éléments affichés en jeu et leur coordonnées. La classe possède également d'une autre **HashMap** contenant pour chaque élément les règles qui lui sont associées. Enfin, des entiers *xMax* et *yMax* représentent la taille de la table de jeu. Une **HashMap** est préférable ici à la représentation pleine d'un tableau à deux dimensions. On ne stocke ici seulement que les cases contenant un élément. Cela économise de la mémoire.
- Une classe **BoardDisplay** qui gère le chargement des images et l'affichage des éléments. Elle contient toutes les données numériques nécessaires pour placer des images à la bonne taille au bon endroit, à savoir les dimensions de la fenêtre qui sert d'interface graphique, et les mêmes *xMax* et *yMax* que **Board**. Ce dernier point est par ailleurs problématique car cela provoque des problèmes dans le bien-être de l'encapsulation du programme.
- Une classe **RuleManager** qui se charge de gérer les règles courantes dans le jeu afin de correctement les actualiser.
- Une classe **SaverLoader** directement inspirée du dernier TP du semestre, qui se charge d'enregistrer ou de charger un **Board** dans un fichier texte. Cette classe permet notamment de charger différents niveaux, et potentiellement de sauvegarder une partie en cours. L'idée est aussi de stocker un **Board** d'une manière moins coûteuse qu'une classe stockant des **Board** pour chaque niveau.
- Une interface **Element** qui représente tout élément individuellement dans le **Board**. Celle-ci est composée de :
  - L'enum **Item** qui représente tous les éléments autre que des mots. Ils peuvent servir de décor ou d'objet interactif. On y retrouve notamment *Baba, Flag , Rock* ou encore *Wall*.
  - L'interface par héritage **Word** qui représente en toute logique les mots dans le jeu. Celle-ci est elle même composée directement de **Operator** qui regroupe tous les mots de type opérateurs dans le jeu (*IS* par exemple), et indirectement de **Name** et de **Property** qui représentent respectivement les noms associés aux **Item**, ainsi que les propriétés telles que *You, Win* etc, qui sont toutes deux dans l'interface **Rule** qui implémente **Word** et qui représente les deux types de règles associées à un élément.

- Un enum **Type** qui regroupe plus simplement les quatre types distincts d'enum dans **Element : Name, Operator, Property** et **Item**. Il permet de connaître le type d'un élément, notamment dans la **HashMap** de **Board**.

Choix et améliorations depuis la soutenance Bêta :

Depuis le 15 décembre, nous nous sommes focalisés sur ce projet. Nous n'avions que peu avancé avant cela, arrêtés à un affichage « sapin de Noël » et une base fragile. Nous avions une bonne idée d'à quoi ressemblerait **Board**, **Item**, et le **BoardDisplay**. **Item** avait évidemment vocation à être représenté plus globalement sous l'interface **Element** avec ses compères de type **Word**. Nous avions même divisé chacun d'entre eux avec une classe associée (exemple pour **Item, ItemType**) avant de finalement fusionner le tout.

Nous avons énormément changé l'architecture, plusieurs fois sûrement de part notre manque d'expérience dans le langage, mais nous pensons être arrivé sur une base satisfaisante, avec l'interface Element qui regroupe tous les différents type d'objets que nous pouvons rencontrer, ce qui facilite la gestion d'éléments précis dans le **Board**.

Nous avons également ajouté le concept de règles, qui après réflexion, nous est apparu comme une **ArrayList** de propriétés dans chaque élément, mais il nous est apparu que la version actuelle (une **HashMap** dans **Board** contenant toutes les règles groupées par élément) était plus simple.

Globalement, tout ce que nous présentons ici est fait après la première soutenance. Nous sommes arrivé à un résultat satisfaisant, mais malheureusement inachevé. En effet, nous n'avons pas eu le temps de faire tous les niveaux, ni de faire la gestion des arguments en ligne de commande, ou encore le niveau extra, et les fichiers .jar et .xml.. Le principal problème du programme est le fait que les éléments du **Board** appellent la méthode ***loadImage*** qui a besoin de connaître les dimensions de la fenêtre et du **Board**, d'où l'obligation de mettre ces données dans le **BoardDisplay**. Cela crée des dépendances cycliques, car nous aimerions obtenir *xMax* et *yMax* dans le **BoardDisplay** qui doit donc être chargé avant le **Board**. Or, l'inverse serait préférable. La solution serait probablement de séparer les données entre **BoardDisplay** et une nouvelle classe.