Projet Approche Objet

UN JEU DE PLATEAU LABYRINTHE

Guillaume CHARLET
Kenji FONTAINE
Romain ORDONEZ
Adrien HALNAUT

22 décembre 2017

Table des matières

1 Description du projet

Ce travail a été réalisé dans un cadre universitaire, par des étudiants en Master 1 informatique à l'université de Bordeaux. Le but de ce projet est de mettre en pratique les connaissances acquises lors du cours d'Approche Objet.
Labyrinth est un jeu de plateau dans lequel le joueur peut se déplacer et devra trouver la sortie dans un labyrinthe en franchissant divers obstacles.

2 Architecture du code

Le dossier src contient notre code source pour le projet, réparti dans différents sous-dossiers. On y trouve également Main, qui est éxécuté lorsque le programme est lancé.

2.1 Main

Description de Main

2.1.1 controllers

Master : Description de Master.

ingame :
— EntitySpawner : Instancie les entités du dossier drawable avec une position aléatoire.
— EventManager : S'occupe de détecter et de gérer les collisions entre entités (entre le joueur et un ennemi par exemple).
— GameManager : Gère le déroulement du jeu. Initialisation, gestion des déplacements des entités, ainsi que leur destruction (si un bonus est ramassé par exemple).
— Inputs : Reçoit les entrées utilisateurs afin de déplacer le joueur dans le labyrinthe.
— SpriteManager : Permet de récupérer les images utilisées dans l'interface graphique.

2.1.2 models

drawable :
— Bonus : Classe des bonbons ramassables par le joueur.
— Character : Classe utilisée par le joueur ainsi que les ennemis. On distingue le joueur des ennemis à l'aide d'un booléen.
— Door : Classe de la porte de sortie utilisée dans le labyrinthe.
— Entity : Classe abstraite héritée par toutes les autres entités. Elle définie le fonctionnement générale d'une entité.
— EntityType : Enumération des différentes entités pouvant être utilisées dans le jeu.
— OnOff : Classe des boutons permettant d'activer/désactiver un mur dans lebyrinthe.
— SpriteType : Enumérations des différentes images pouvant être associées à une entité dans le jeu.

game :
— maze/Direction : Enumération des directions différentes qu'un mouvement peut prendre.
— maze/Menu : Classe du Menu, non utilisée.
— maze/WallType : Enumération des différents états qu'un mur peut prendre.

2.1.3 views

ViewFrame : Permet d'initialiser l'interface graphique ainsi que de mettre l'affichage des entités à jour à chaque déplacement.

3 Description du travail

3.1 Première séance

Lors de la première séance, nous avons réfléchi sur la structure qu'allait prendre notre code. Nous nous sommes mis d'accord pour partir sur une architecture MVC (modèle, vue et contrôleur) afin de disposer d'une certaine flexibilité. Nous avons ensuite déterminer sur papier, la hiérarchie de nos classes, packages, etc... Puis nous nous sommes réparti le travail en groupe.
Voici le premier modèle retenu :

- Model
  - Drawable
    - Gentil
    - Mechant
  - Static
    - Switches
    - Doors
    - Bonus
  - Game core
    - Laby/Graph
    - Menu
- View
  - Rien pour l'instant
- Controller
  - Main/Master
  - In-Game
    - Inputs
    - Events
    - GameManager

3.2 Premier pas

Nous avons commencé notre implémentation par la génération de graphe (modèle) ainsi que l'affichage de la grille (vue). Le but était d'obtenir une interface graphique avec un affichage d'un labyrinthe. Pour cela, il nous a également fallu implémenter Master (contrôleur) afin de faire le lien entre le graphe généré en modèle et l'affichage. Cette partie nous a pas posé de problème particulier. Guillaume s'est occupé de la partie contrôleur, Adrien de la génération de graphe et l'affichage a été fait par Romain et Kenji.

3.2.1 Modèle graphe

Si tu peux remplir cette partie Adrien.

3.3 Implémentation des entités

3.3.1 Déplacement du joueur

Une fois la génération et l'affichage du labyrinthe effectués, notre prochain objectif était d'implémenter le joueur (affichage + déplacement). Pour se faire, nous avons commencé par implémenter une classe abstraite Entity, afin de ne pas avoir de redondance dans notre code. Puis nous avons crée une classe Character héritant de Entity, en implémentant les quelques fonctionnalités propres à cette classe (comme le déplacement).

3.3.2 Objets statiques

L'implémentation des objets statiques que l'on peut afficher tel que les bonbons a été faite de façon naturelle. Il nous suffisait d'hériter de la classe Entity et d'implémenter les quelques fonctionnalités propres à chaque objet. Sauf pour les boutons OnOff permettant d'activer/désactiver une porte, étant lié à un certain mur, il était plus compliqué d'implémenter ces boutons.

3.3.3 Gestion des collisions

Le prochain objectif était de faire disparaitre les bonbons au contact du joueur. Pour cela nous avons créer un contrôleur EventManager dont le but est uniquement de gérer les collisions. EventManager parcourt la liste de toutes les entités en jeu et envoie un signal aux entités si il y a collision.

3.3.4 Ennemis

Partie pour Romain.