Université Gustave Eiffel

BERNIER Aurélien
BLONDEL Lucas

Projet Java : Manuel dev

Sommaire

I. Introduction

Objectif

Le but de ce projet était de créer en Java un jeu de type aventure où l'on peut créer ses propres cartes d'aventure pour ensuite pouvoir les jouer. Le principe du jeu se rapproche plus d'un jeu où l'on créée sa propre histoire.

Le jeu se base sur la bibliothèque Zen5 et sur les tuiles du jeu « Baba Is You ».

Avancement du projet

Nous avons réalisé toute la phase 0 qui était le minimum requis et une partie de la phase 1. En détails nous avons implémenté :

Phase 0 : Affichage de la map avec éléments décoratifs, obstacles, ennemis et épée, le système de points de vie (un joueur prend des dégâts s'il est devant un ennemi) et les ennemis avec un comportement stroll.

Phase 1 : Système de points de vie avec nourriture (sans la nourriture à cuire), centrage de la carte sur le joueur et scroll, les armes stick et sword et un réel inventaire pour le joueur.

Il faut noter que bien que le lexer fonctionne, nous n'avons pas réussi à faire en sorte qu'il indique toutes les erreurs contenues dans une carte mais seulement la 1ère.

Répartition des tâches

Pour le projet, Aurélien s'est occupé de réaliser le lexer/parser pour pouvoir valider et traduire les cartes dans la structure du projet.

Lucas a quant à lui réalisé la partie moteur du jeu et la partie graphique/interactions joueur-jeu.

Nous également avons conçu toutes les structures du moteur à deux.

II. Architecture

Packages

Nous avons divisé le code en plusieurs packages :

- **fr.uge.cmdline** : partie du code qui sert à analyser la ligne de commande du lancement du jeu avec les options possiblement entrées (--level, --dry-run ou --validate)

- **fr.uge.graph** : la partie affichage des éléments et traitements des actions du joueur et de son inventaire

- **fr.uge.lexer** : c'est ici que se trouve les classes nécessaires au lexer/parser de map du jeu pour les valider

- **fr.uge.main** : là où se trouve la classe Main du programme

- **fr.uge.thebigadventure** : l'ensemble des types de base du programme en plus du code nécessaire au moteur du jeu

Classes et Record

Analyse ligne de commande

Dans le package correspondant il n'y a qu'une seule classe pour l'analyse de la ligne de commande : CmdLine, contenant un champ cmdline qui est une HashMap<String, String> où la clé représente l'option de la ligne et la valeur l'argument correspondant à cette options sachant que l'option –dry-run ne possède pas d'argument et donc a une valeur null.

Dans l'ordre, la ligne est analysée (chaque option est ajoutée à la HashMap), puis on vérifie chaque options en plus de la présence de –level et enfin on renvoie le chemin de la map pour l'envoyer au lexer.

Analyse carte

Les classes/record Lexer, Token et Result nous ont été fournies par M.Forax et permettent d'analyser lexicalement un fichier de carte.

Nous avons conçu la classe GameFromFile qui permet d'extraire les informations du fichier .map pour les convertir en structures de données du programme et aussi d'indiquer les potentielles erreurs contenues dans le fichier.

Le package contient aussi des enum pour les décorations, ennemis, obstacles et nourriture pour pouvoir vérifier si les skin précisés sont les bons et pour pouvoir attribuer le bon type aux éléments.

Moteur

Contenu dans fr.uge.thebigadventure, c'est le package qui contient le plus de classes et qui défini les structures de base pour le fonctionnement du jeu.

Pour le mouvement et la position des éléments, nous avons utilisé un enum pour la Direction et un record Position pour la position. Une position est définie par ses coordonnées x, y. Par exemple un joueur avec une position Position(1, 2) signifie qu'il est sur la case de colonne 1 et ligne 2.

L'interface Tile rassemble les records Obstacle et Decoration qui représente les obstacles et décorations du jeu. Ces deux records n'ont que le champ skin.

Tile est notamment utilisée dans la classe Grid qui représente la grille du jeu avec ses décorations et obstacles. Cette classe contient aussi un champ grid qui est une HashMap<Position, Tile> où les clés sont les positions des Tiles et les valeurs les tiles en elles mêmes. Cette structure nous permet ainsi de retrouver une tile par sa position.

Les items du jeu sont divisés en 3 types : Weapon pour les armes, Nutrient pour la nourriture et Exchangeable pour les autres éléments qui étaient censés êtres échangeables et qui ne sont ni de la nourriture ni une arme. Ces items sont représentés par des records avec les champs name, skin et damage pour les armes.

Les items sont tous rassemblés sous l'interface Item utilisée pour la classe ItemGrid qui représente les items sur la grille. Tout comme Tile, le champ items est une HashMap<Position, Item> où à une clé Position correspond une valeur Item.

Les ennemis sont représentés par le record Enemy où sont utilisé la Direction, la Position et une Zone qui est un record représentant la Zone où peut se déplacer l'ennemi.

Enemy est utilisé dans EnemiesGrid où tout comme ItemGrid ou Grid, on utilise un HashMap<Position, Enemy>. EnemiesGrid est une classe qui représente les ennemis présents sur la carte.

Le joueur est représenté par la classe Player qui contient des champs similaires à Enemy mais avec un champ holdItem qui représente l'item dans la main du joueur et Inventory qui est l'inventaire du joueur.

Inventory est une classe pour l'inventaire du joueur contenant un champ selection qui est l'item sélectionné par le joueur pour sa main et inventory qui est une HashMap<Item, Integer> où les clés sont les items trouvés et les valeurs la quantité. Ainsi, si le joueur trouve par exemple 3 bananes, il aura dans son inventaire 3 bananes sur la même case.

Pour Enemy et Player, nous avons opté pour des classes plutôt que des records car il nous fallait des champs modifiables pour la vie par exemple.

Enfin, Game est une classe qui permet de manipuler tout le jeu. C'est ici qu'est stocké dans ses champs, les classes ItemGrid, EnemiesGrid, Grid et Player et que la boucle principale du jeu est mise en œuvre dans la méthode mainLoop où sont traités dans l'ordre les attaques des ennemis, les items que le joueur peut ramasser sur la carte, les actions claviers et enfin l'affichage.

Graphique

Dans le package fr.uge.graph, il y a deux classes . D'abord la classe GraphicInterface qui permet d'afficher les éléments de la carte sur la fenêtre, de calculer la taille des tiles en fonctions de la taille de fenêtre et de calculer le scroll de la caméra centrée sur le joueur.

La classe contient un champ context de type ApplicationContext qui sert à dessiner les éléments de la fenêtre et un champ imagesGame qui est une HashMap<String, BufferedImage> où les clés sont les noms des images et les valeurs les images associées à ce nom. L'intérêt de ce dernier champ est de charger toutes les images nécessaires à la map courante dans la HashMap à partir de la méthode loadImage et de pouvoir ensuite accéder à ces images depuis n'importe quelle méthode de dessin via un get sans devoir charger les images à chaque fois que l'on a besoin de les afficher.

Ainsi dans l'ordre, toute les images sont chargées puis on dessine chaque élément du jeu en accédant aux images contenues dans la Map.

La 2ème classe du package est PlayerActions qui gère les actions du joueur. Plus spécifiquement, cette classe contient une méthode pour convertir les touches flèches en directions, une méthode pour changer l'item sélectionné de l'inventaire et une méthode pour gérer les actions de l'inventaire tant qu'il est ouvert.

III. Améliorations depuis la soutenance

Remarques faites à la soutenance

Lors de la soutenance notre correcteur nous a dit que dans l'ensemble le projet était bien construit avec cependant quelques petit détails à changer :

- faire en sorte que le joueur ne traverse pas les ennemis et inversement

- dédier un package pour la partie graphique

- améliorer le lexer/parser en le découpant en plusieurs méthodes

Ajouts réalisés

Après la soutenance nous avons donc ajouté au projet :

- une correction du lexer en le divisant en plusieurs méthodes

- la création des packages fr.uge.graph et fr.uge.cmdline

- la corrections des collisions ennemis-joueur

- l'affichage des noms des entités et d'un petit écran de fin de partie

- la nourriture sans les éléments à cuire

- le scroll de la map

- l'inventaire du joueur avec les actions correspondantes dans le package fr.uge.graph

- un package dédié à l'analyse de la ligne de commande de lancement du jeu avec en plus les options –level, --dry-run et –validate

- ajout du type exchangeable pour les items qui ne sont ni des armes ni de la nourriture

- du nettoyage de code et corrections de quelques bugs

IV. Conclusion

Pistes d'ajouts/améliorations

Bien que nous ayons terminé la phase 0 et la moitié de la phase 1, nous aurions voulu ajouter le système de clés et portes et les système d'amis avec les échanges. Cependant à cause des Projets de C et Analyse Syntaxique et des révision pour les examens nous avons manqué de temps pour tout ajouter.

De plus, comme précisé plus tôt le lexer/parser de cartes fonctionne mais il n'indique que la 1ère erreur d'un fichier et pas toutes les erreurs car nous n'avons pas réussi à trouver comment faire.

Difficultés

Le plus difficile dans le projet a été de trouver les bonnes structures et aussi faire le parser de cartes pour pouvoir les valider ou non.

Apports

Si le projet reste assez dur et lourd tout de même, il reste très intéressant à faire pour une 1ère année de Java et nous a poussé à aller un peu plus loin que ce que nous faisions auparavant.

Même si nous n'avons pas pu tout faire, nous avons essayé de faire le maximum et de bien le faire avec les contraintes de temps imposées et les deux autres projets en C et Analyse. Nous aurions espéré faire plus mais nous sommes tout de même très content de ce que nous avons pu produire.