Rapport du projet informatique 2023
"Road to Master"

Debucquoy Anthony Matteo Di Leto

May 2023

Contents

Introduction

Lors de ce deuxième quadrimestre, le projet Informatique proposé par notre université fut partie intégrante de notre emploi du temps. Régulièrement, nous nous sommes rassemblés pour nous organiser et trouver une direction dans laquelle nous voulions voir notre projet évoluer. Grâce aux objectifs fixés par nos enseignants, nous sommes - nous le pensons - maintenant plus aptes à nous confronter à ce genre d'objectifs, tant au niveau personnel qu'en tant que groupe. Il va sans dire que comme pour tout projet, notre chemin a été semé d'embûches. En l'occurrence, nous souhaitons faire part de l'abandon d'un de nos membre. Eddy Jiofak qui souhaite se réorienter. Nous lui souhaitons une bonne reconversion.

Objectifs

Voici l'objectif fixé par nos enseignants. (document de consignes)

> Le but final de ce projet est de réaliser une application graphique en Java permettant de jouer au jeu "Cats Organized Neatly". Ce jeu est un jeu de type "puzzle" où le joueur doit placer des pièces de formes différentes pour combler l'aire de jeu. L'application devra permettre de sauvegarder et charger une partie et de créer des niveaux automatiquement.

1 Organisation

Lors de nos rassemblement pour le projet, toutes les idées émises se sont retrouvés sur un blog afin de pouvoir y accéder de manière efficace. Ce blog nous sert également à garder une trace de l'évolution du projet.

1.1 Choix

- le VCS `git` pour garder une trace de l'avancement du projet. Avec comme remote une instance privée de gitea nous permettant de vérifier les MR/PR plus efficacement.
- Une instance de DroneCI permettant de vérifier que le projet soit toujours compilable et que les tests ne soient pas ratés sans que nous nous en rendions compte.
- Javafx, comme recommandé par nos enseignants.
- Les pièces et niveaux sont stockées sous forme de matrice de booléen
- Un parser de fichiers efficace et donnant des fichier légers.

Shapes

Les pièces ainsi que la carte sont représentés par une matrice de booléen `boolean[][]`. Nous avons donc une classe `shape`, parent de `Map` et de `Piece` dans lequel nous stockons notre matrice. Ensuite, `Map` Contient une liste de `Piece`. Ces pièces contiennent une position représentée par la classe `Vec2`. Cette position est la position du carré supérieur gauche dans la `Map`. Avec toutes ces informations nous avons le nécessaire pour le moteur du jeu.

Il est facilement possible de manipuler la carte et les pièces. Il nous suffit alors de faire correspondre l'affichage avec ces différentes classes. Ce qui est entrepris par la classe `GameUI`.

Le tout est géré par la classe `Controller` qui permet de choisir entre l'affichage d'un menu ou d'une partie en cours.

2 Points Forts

2.1 Parser de fichiers

Pour la rétention des niveaux, plusieurs possibilités s'offraient à nous. Nous avons alors décidé d'accomplir une série d'objectifs propres à notre projet

avec un parser de fichiers dédié. Nous voulions que ce parser accomplisse les objectifs suivants:

- Les données du niveau seront encapsulées dans un header/footer pour laisser la possibilité d'enregistrer plus d'informations (images/musiques) dans un seul fichier dans le futur.
- La taille du fichier devra être aussi petite que possible, tout en gardant les informations nécessaires au bon fonctionnement du jeu.
- Il sera possible d'enregistrer l'état d'une partie en cours.

Ce parser est implémenté par la classe `BinaryParser`.

spécification

Header/Footer Les données du niveau commencent par les 3 *caractères* 'S', 'M', 'S' (ou `0x534D53`) et se terminent par les 3 *caractères* 'S', 'M', 'E' (ou `0x534D45`)

Taille de carte Le premier octet des données représente la largeur de la carte, le second sa hauteur.

Forme de la carte Chaque cellule de la carte est représenté par un 1 ou un 0. Le 1 représente un emplacement libre, un 0 une cellule vide. La forme de la carte peut alors être répartie sur un nombre indéterminé d'octets. Nous pouvons déterminer ce nombre grace à

$$\frac{\text{largeur} * \text{hauteur } (+1 \text{ si multiple de } 8)}{8}$$

en division entière.

Nombre de pièce L' (les) octet(s) qui forme(nt) la carte représente le nombre de pièce

Pour chaque pièces

Taille de la pièce La taille est représentée sur un seul octet sous forme de nibble[1]. La première partie du nibble est la largeur. La seconde partie du nibble est la hauteur

Forme de la pièce Chaque cellules de la pièce est représentée par un 1 ou un 0. La manière de le représenter et exactement la même que pour la forme de la carte

[1] https://en.wikipedia.org/wiki/Nibble

Dans le cas où le fichier sauvegarde l'état de la partie, à la fin, et pour chaque pièce dans le même ordre que l'apparition des pièces:

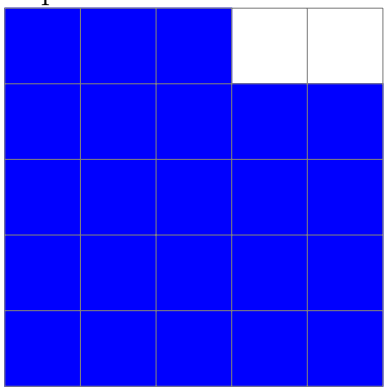
Position de la pièce 2 octets par pièces, 1 octet pour sa position en x et 1 octet pour sa position en y. Dans le cas où la pièce est flottante (n'est pas placée dans la carte.), les octets contenant les caractères F puis L (0x464C) remplacent les coordonnées

Exemple

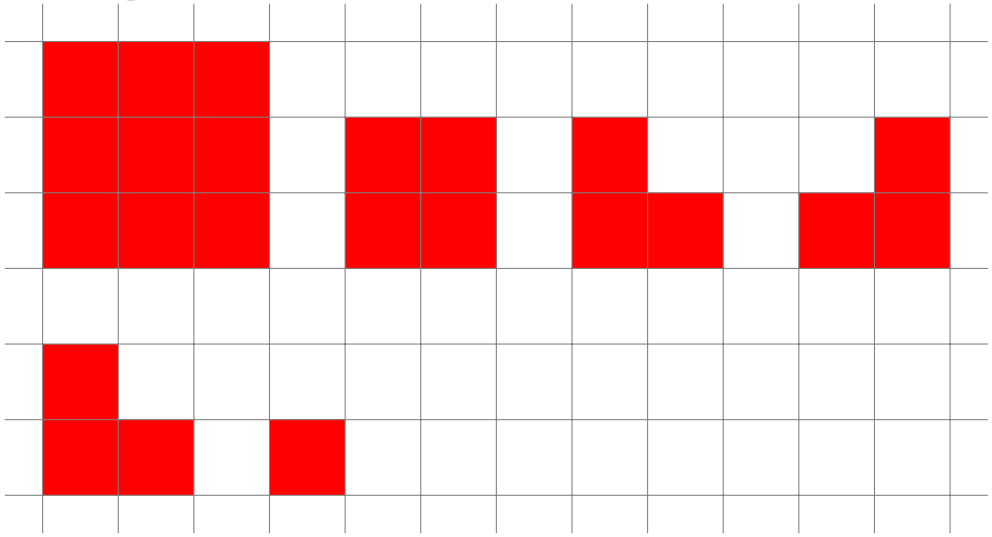
Voici le 'hexdump' du niveau 11

```
53 4d 53 05 05 e7 ff ff  80 06 33 ff 80 22 f0 22  |SMS.......3.."."|
b0 22 70 22 b0 11 80 53  4d 45                    |."p"...SME|
```

Représente une carte de la forme :



Avec les pièces :



En plus de ce parser, et dans le cas où ce premier ne serait pas capable de stocker certaine carte (par exemple si une pièce mesure plus de 15x15), nous avons également implémenté un parser très simple en utilisant l'interface `Serialize` de java. Ce parser est implémenté et fonctionnel, mais n'est pas utilisé dans le projet à l'heure actuelle.

Ces deux parseurs implémentent l'interface `FileParser`.

Finalement, La classe `FileParserFactory` permet une utilisation simple de ces parseurs. Celle-ci contient deux fonctions statiques.

- `Map loadMapFromFile(File)` permet de retourner la Map d'un fichier chargé.
- `void saveFileFromMap(File, Map)` permet de sauvegarder une map dans un fichier.

Dans le cas d'une sauvegarde ou d'un chargement, le parser est choisi en fonction de l'extension de fichier ('.level', '.slevel', .'serialized', '.sserialized').

L'avantage de ce système est que nous pouvons facilement ajouter d'autres parser de fichiers dans le futur.

2.2 Générateur de niveaux

Le générateur de niveaux a été conçu de sorte à proposer 3 difficultés différentes:

- Niveau Facile
- Niveau Moyen
- Niveau Difficile

L'algorithme derrière est le même. En voici le principe :

Gestion du plateau

Le joueur choisit une difficulté. En fonction de la difficulté choisie, la grandeur du plateau de jeu sera différente. Si la difficulté choisie est facile ou moyenne, alors un curseur parcourt les extrémités du niveau. Ce curseur sélectionne aléatoirement les cellules qui seront gardées ou non. Grâce à ça, la forme du plateau n'est pas trop carrée.

Nous nous sommes basés sur le même principe pour le niveau de difficulté difficile mais en plus d'une taille encore plus grande, le curseur parcourt les extrémités avec une profondeur de 2 afin de faire varier la carte plus grande. Cela introduit le problème des cases isoléss. C'est pourquoi une boucle vérifie chaque cellule et la supprime si celle-ci est isolée.

Gestion des pièces

Peu importe la difficulté du niveau, voici le fonctionnement :

Une taille maximum des pièces a été fixée au préalable à 3x3. Par la suite, un curseur parcourt des cases de la carte préalablement conçue de manière aléatoire. Pour chaque case, l'algorithme teste pour chaque case de la pièce, si l'espace est disponible. Si ça n'est pas le cas, alors la pièce est modifiée afin de faire correspondre la pièce et la carte.

L'avantage de cette méthode est que les niveaux sont tous très différents. Les désavantages sont que, par malchance, il est possible d'avoir énormément de piece 1x1. il est aussi plus difficile d'appliquer des textures et dessins - à l'image du jeu de base - sur les pièces.

Malgré tout, avec nos nombreux tests, ce générateur de niveaux nous satisfait vraiment bien et la difficulté des niveaux correspond bien aux attentes.

2.3 Interface graphique

L'interface graphique du jeu tient sur 5 classes différentes.

Controller

Classe principale. Elle s'occupe de la gestion des autres classes et de la cohérence entre elles. Toutes les autres classes (présentes dans le package `Scenes`) sont des sous classe de Parents. Cela permet de les afficher grâce à la méthode statique `switchRoot`. C'est aussi le point d'entrée du programme.

MenuAccueil

Classe s'occupant de générer la page d'accueil du jeu. C'est-à-dire la première page que verra l'utilisateur. Cette page permet d'accéder aux niveaux du jeu de base, mais également de générer les niveaux aléatoires. De plus un dernier bouton "Load Game" permet de revenir sur la dernière partie en cours du joueur.

MenuLevel

Classe s'occupant d'afficher les niveaux proposés dans le jeu, et qui, en fonction du jour choisi, change les boutons et les niveaux disponibles.

ScreenLevelFinish

Classe qui s'affiche à l'écran dès que le joueur a fini le niveau. Celle-ci propose également de réessayer le niveau ou de retourner au menu principal.

GameUI

Classe s'occupant de l'affichage d'un niveau. S'occupe dans un premier temps d'afficher le plateau au milieu de l'écran. Par la suite, les pièces sont dispersées à la gauche de l'écran et sont rendues disponibles à l'utilisateur. Ces pièces sont déplaçables à l'aide de la souris. Si elles sont lâchées sur un emplacement disponible du plateau, alors elles vont s'aligner avec ses cellules. Pour se faire, nous regardons si le centre d'une cellule (la plus en haut à gauche) est présente dans une cellule du plateau. Ensuite, on vérifie que toutes les cellules de la pièce ont bien la place dans le plateau pour se poser. Si c'est le cas, alors la pièce est posée.

3 Points Faibles

Bien que l'interface graphique permet de naviguer de manière fluide dans le jeu, il y a clairement un manque d'animation et de dynamisme *i.e*: transition de page à une autre, apparition des boutons ainsi qu'un manque de musique. Toujours concernant l'affichage l'adaptation à la taille de l'écran peut être revue.

De plus suite à la perte de notre membre nous n'avons pas su gérer la partie du projet qui concerne le design/textures des pièces ainsi que l'histoire jeu préparée auparavant.

4 Apports Positifs et négatifs

4.1 Anthony

Personnellement, ce projet m'a permis de me plonger dans la conception d'un format de fichier personnalisé. C'est un exercice que je n'avais pas encore fait jusqu'a maintenant. Malgré mes efforts pour prévoir un maximum de choses à l'avance afin d'éviter de devoir modifier ma spécification pendant le développement, je me suis vite rendu compte que je n'avais pas pensé à tout et que je devrais changer des étapes pour pouvoir arriver à mes fins. Je pense que ce parser de fichier est vraiment améliorable mais je suis relativement fier du résultat.

J'ai pu présenter ce parser à Dr Quoitin qui a pu me conseiller sur différentes approches à ce problème. J'en prends bonne note.

4.2 Matteo

Il est clair que je peux tirer plusieurs enseignements grâce à la réalisation de notre projet. Tout d'abord, j'ai pu en apprendre beaucoup plus concernant la P.O.O en java, mais aussi j'en ai appris d'avantage sur l'utilisation de la bibliothèque JavaFx.

De plus, durant la réalisation du projet, comme dit précédemment, nous avons utilisé plusieurs outils dont la plus grande découverte pour moi est Git qui révèle son intérêt pour les travaux de groupes et qui, selon moi, se révélera tout aussi essentiel pour mes futurs projets scolaires ainsi que professionnels.

5 conclusion

En conclusion, nous pouvons séparer notre travail en trois parties différentes:

1. Le parser de fichier (gestion sauvegarder/charger partie)
2. Logique du jeu (Map,Vec2,Shapes)
3. Liaison à l'UI (Javafx)

Malgré notre travail concentré sur le bon fonctionnement du jeu avec un parser suivant nos objectifs, une utilisation de la P.O.O de manière très efficace ainsi qu'une approche correcte de l'utilisation du framework Javafx, d'autres améliorations sont toujours possibles! En effet, l'idée d'ajouter une histoire, des trophées, un Easter egg, des pièces spéciales ou un encore une table de score basée sur le temps, reste possible afin de rendre notre jeu encore plus complet.

En conclusion, notre jeu a encore plein de possibilité afin d'être encore plus complet et amusant!