Rapport du projet informatique 2023
"Road to Master"

Debucquoy Anthony Matteo Di Leto

May 2023

Contents

Introduction

Lors de ce deuxième quadrimèstre, 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 - je le pense - maintenant plus apte à nous confronter à ce genre d'objectifs. Tant au niveau personel qu'en tant que groupe. Il va sans dire que comme pour tout projets, notre chemin a été semé d'embuches. En l'occurence, 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.

1.2 Difficultés

2 Points Forts

2.1 Parser de fichiers

Pour la rétention des niveaux, plusieurs possibilités s'offraient à nous. Nous avons alors décidés d'accomplir une série d'objectifs propre 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 nécessaire 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 Chaques cellules 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 chaques 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 Chaques 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

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

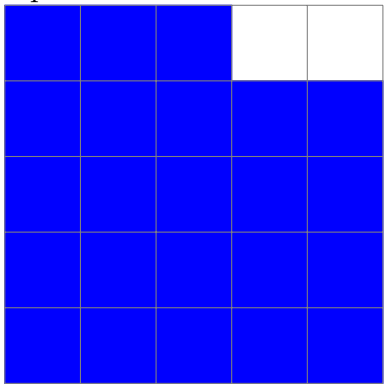
Position de la pièce 2 octets par pièces, 1 octet pour sa position en x et 1 octet pour sa position en y. Dans le cas où la pièce est flotante (n'est pas placée dans la carte.), les octets contenant les caractères F puis L (0x464C) remplacent les coordonées

Exemple

Voici le 'hexdump' du niveau 11

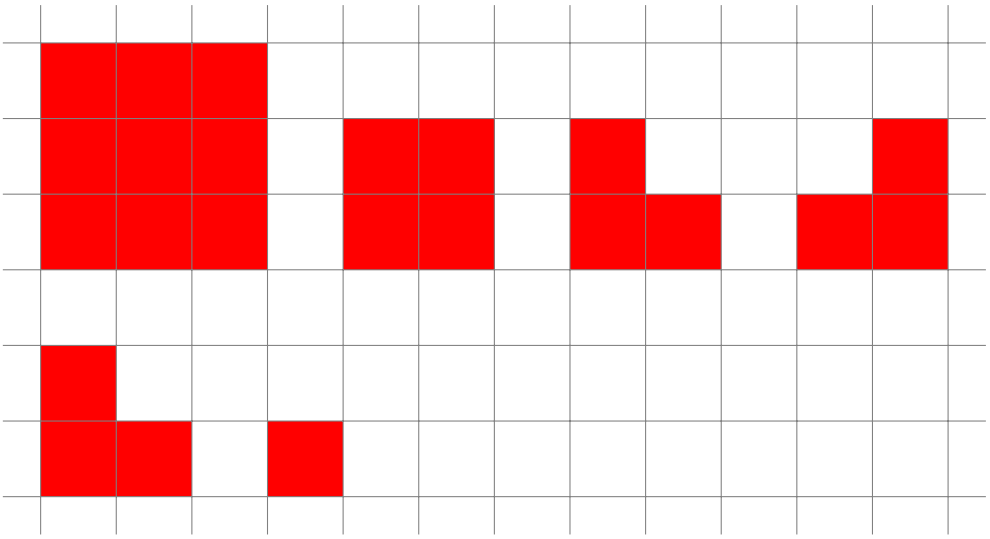
```
53 4d 53 05 05 e7 ff ff  80 06 33 ff 80 22 f0 22  |SMS.......3.."."|
b0 22 70 22 b0 11 80 53  4d 45                    |."p"...SME|
```

représente une carte de la forme :



avec les pièces :

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



En plus de ce parser, et dans le cas où ce premier parser 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` afin de pouvoir utiliser l'un où l'autre parser.

Finalement, La classe `FileParserFactory` permet une utilisation simple des 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 system est que nous pouvons facilement ajouter d'autres parser de fichier dans le futurs.

2.2 Générateur de niveaux aléatoire

Le générateur de niveaux a été conçue de sorte à proposé trois sorte de niveaux différents :

- Niveau Facile
- Niveau Moyen

- Niveau Difficile

L'algorithme derrière est le même en voici le principe :

Gestion du plateau

Le joueur choisi une difficulté en fonction de la difficulté choisie la grandeur du plateau de jeu sera différente. Si la difficulté est facile l'algorithme s'arrête la. Par contre si la difficulté choisie est moyen alors en plus de générer un map plus grande, un curseur parcours les extrémités du niveau et mets de manière aléatoire des trous dedans. Nous nous sommes basé sur le même principe pour le niveau de difficulté difficile mais en plus d'une taille encore plus grande, le curseur parcours avec une profondeur de deux afin de mettre des trous de la map. Cela à inclus au code une vérifiction afin de supprimer toutes les cases isolées.

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 parcours le niveau créer en commençant en haut à gauche puis part dans 8 directions différentes. (en haut à gauche, en haut au centre,en haut à droite, à droite au centre, en bas à droite, en bas au centre, en bas à gauche et au centre à gauche) Ce qui explique la taille maximum de la pièce. Si le curseur rencontre un obstacle(trou dans le niveau ou espace déjà utilisé pour une autre pièce) alors la pièce est modifié (comporte un 0 (false) à l'endroit en question) si pas l'espace est rajouté sur la pièce.

2.3 Fluidité et Simplicité de l'interface grahique

En effet, tout l'affichage du jeu tient sur 5 fichiers différents

Le controller

qui s'occupe d'afficher chaque scènes différentes en fonction des actions du joueur.

Le MenuAccueil

qui s'occupe de générer la page d'accueil du jeu c'est-à-dire la première page que vera l'utilisateur qui permet d'accéder aux niveaux de base proposé par le jeu Cats Organized Neatly,mais aussi de pouvoir lancer un niveau généré

aléatoirement en fonction du niveau de difficulté choisi. De plus un dernier boutons "Load Game" permet de récupérer directement le niveau lorsque le joueur l'a quitté.

Le MenuLevel

qui s'occupe d'afficher les niveaux proposés dans le jeu et qui en fonction du jour choisir change tout les boutons et les niveaux disponibles.

ScreenLevelFinish

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

Le GameUI

qui s'occupe de placer le plateau de jeu au milieu de l'écran ainsi que d'afficher les pièces à sa gauche, mais aussi de gérer le dépassement pour ne pas que les pièces soient affichées en dehors de l'écran.

3 Points Faibles

- L'interface Bien que l'interface graphique permet de naviguer de manière fluide dans le jeu, il manque clairement un manque d'animation et de dynamysme ie : 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 revu.
- Le design

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

4 Apports Positifs et négatifs

4.1 Anthony

Personellement, ce projet m'a permis de me plonger dans la conception d'un format de fichier personnalisé. C'est une chôse que je n'avais pas encore fait jusqu'à maintenant. Et malgré mes efforts pour prévoir un maximum de

choses à l'avance afin d'éviter de devoir modifier ma spécification pendant le dévelopement. Je me suis vite rendu compte que je n'avais pas pensé à tout et que je devrais changer des choses 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 prend bonne notes.

4.2 Matteo

Il est claire que je peux tirer plusieurs enseignement 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 bibliotheque JavaFx.

De plus, durant la réalisation du projet, comme dit précédemment nous avons utilisé plusieurs outils et dont la plus grande découverte pour moi : Git. Qui révèle son intérêt pour les travaux de groupes, et qui selon moi ce 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. Instanciation du jeu (Map,Vec2,Shapes)
3. Liaison au graphisme (Javafx)

Malgré notre travail fourni au bon fonctionnement du jeu avec un parser des plus efficace, une utilisation de la P.O.O de manière très efficace, ainsi qu'une approche correct avec Javafx, d'autre amélioration sont toujours possible ! En effet une idée de rajouter une histoir, des trophées, un easter egg(plus particulierement un tetris) ou un encore un table de score basé sur le temps reste possible afin de rendre notre jeu encore plus complet.