Rapport du projet informatique 2023
"Road to Master"

Debucquoy Anthony Matteo Di Leto

May 2023

Contents

1 Organisation **4**
1.1 Choix 4
1.2 Difficultés 5

2 Points Forts **5**
2.1 Parser de fichiers 5
2.2 generateur de niveaux 7

3 Points Faibles **7**

4 Apports Positifs et négatifs **7**
4.1 Anthony 7
4.2 Matteo 8

5 conclusion **8**

Introduction

Lors de ce deuxième quadrimèstre, le projet Informatique proposé par l'université fut une 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. Grace 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 rasemblement 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 serait toujour compilable et que les tests ne seraient pas raté sans que nous nous en rendions compte.
- Javafx, comme recommendé par nos enseignants.
- Pieces les Niveaux sont stockées sous forme de matrice de booléen
- Un parser de fichiers efficace et donnant des fichier légers.

Shapes

Les pieces comme le tableau 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 caré supérieur gauche dans la `Map`. Avec toutes ces informations nous avons le nécéssaire pour le moteur du jeu.

Il est facilement possible de manipuler la carte et les pièces. Il nous suffit alors de faire corespondre l'affichage avec ces différentes classe. 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 nécéssaire 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 pièces L'octet suivant l' (les) octet(s) qui forme(nt) la carte represente le nombre de pièces

Pour chaques pieces

Taille de la piece 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 piece est représenté 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 pieces dans le même ordre que l'apparition des pièces:

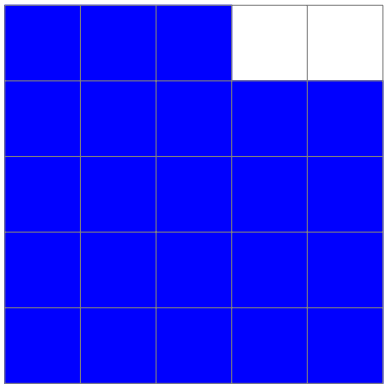
Position de la pièce 2 octets par pieces, 1 octet pour sa position en x et 1 octet pour sa position en y. Dans le cas où la pièce est flotante (n'est pas placée dans la carte.), les octets contenant les caractères F puis L (0x464C) remplacent les coordonées

Exemple

Voici le 'hexdump' du niveau 11

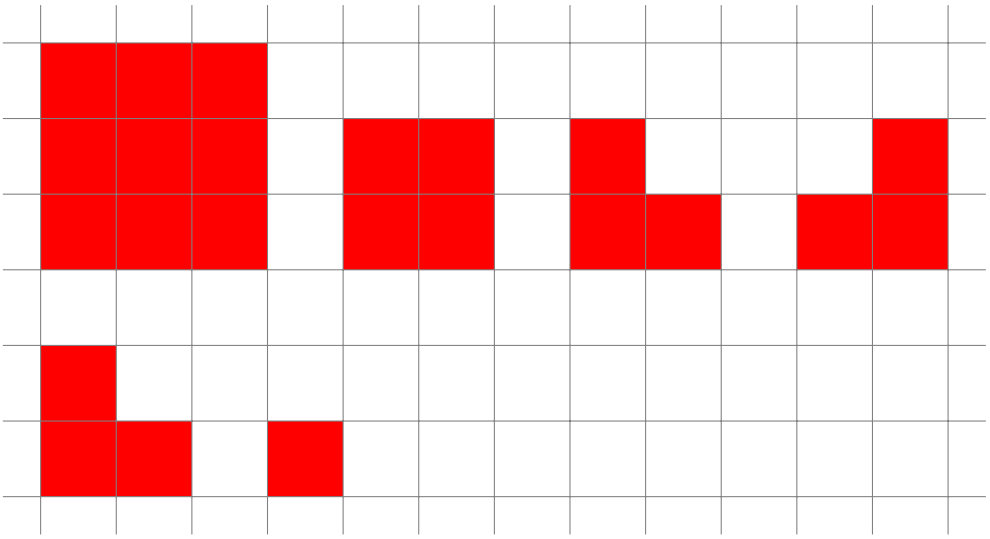
```
53 4d 53 05 05 e7 ff ff  80 06 33 ff 80 22 f0 22  |SMS.......3.."."|
b0 22 70 22 b0 11 80 53  4d 45                    |."p"...SME|
```

représente une carte de la forme



avec les pieces

[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 piece mesure plus de 15x15). Nous avons également implémenté un parser trés 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` affin de pouvoir utiliser l'un où l'autre parser.

Finalement, La classe `FileParserFactory` permet une utilisation simple des parseurs. Celle-ci contient deux fonction statique.

- 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 generateur de niveaux

Le generateur de niveau permet 3 niveaux de difficultés différents.

3 Points Faibles

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'a 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

5 conclusion