

Titanic

- Rapport -

Tristan Savaria et Jonathan Forget

4 novembre 2013

Table des matières

1	Description et utilisation des fonctions	3
1.1	Utilisateur	3
1.2	Programmeur	3
2	Structures de données employées et justification	4
3	Algorithmes et justifications	4
4	Discussion	4
5	Améliorations possibles	5

1 Description et utilisation des fonctions

1.1 Utilisateur

Le programme est une réplique du jeu de table Titanic de la compagnie Smart Games. Le programme charge un fichier texte au lancement et génère une grille de jeu. Le programme est capable d'importer un fichier .txt de format UTF-8 contenant une grille de jeu de dimension 5 par 5. Une ligne vide ou une ligne commençant par # est ignorée. Un tiret (-) représente une case vide, une lettre est un naufragé et un chiffre complété d'une flèche représente un bateau. Les flèches représentent la direction du bateau et les symboles possibles sont \leftarrow \rightarrow \uparrow et \downarrow . Le joueur peut sélectionner un bateau avec un clic gauche de la souris. Une fois sélectionné, les flèches du clavier peuvent être utilisés pour déplacer la bateau dans la même direction sous condition que la destination n'est pas bloqué ou hors grille. Lorsqu'un bateau se déplace à côté d'un naufragé, le naufragé rentre dans le bateau et le navire est désormais impossible d'être déplacé. Le but du jeu consiste à sauver tout les naufragés.

1.2 Programmeur

La classe *ImporteurGrille* prend un int qui deviendra une constante privé pour forcer la dimension de la grille lors de la création de l'objet. La méthode *Chargement* prend un String représentant le chemin vers un fichier .txt. La méthode *lis* lit le flux dans le format UTF-8, ignore les lignes vides ou commençant par le symbole # , rajoute chaque caractère lue dans un tableau de char de deux dimensions et retourne le tableau.

La classe *EcranJeu* est un JFrame responsable de la fenêtre du programme, ainsi des menus disponibles au dessus de la fenêtre. *EcranJeu* instancie la classe *Plateau*, *Plateau* est une composante fille de *EcranJeu*.

Plateau est la grille de jeu de taille fixe. *Plateau* utilise un GridLayout et a des objets de type *Case* comme composantes filles. À partir d'un tableau de char de 2 dimensions, la méthode *GenerateGrid* instancie les objets *Cases*, les positionnes dans un tableau de *Case* de deux dimensions et finalise par rajouter les *Cases* comme composante de *Plateau*. La méthode *SetKeybind* lie un dictionnaire de touche d'entrée du clavier à un dictionnaire d'Action, utilisé pour déplacer les bateaux selon les touches du clavier. Finalement, la classe vérifie si les mouvements des bateaux sont légaux.

Dans la classe *Case*, classe fille de JPanel, plusieurs constructeurs sont disponibles. Pour les case vide sur la grille, le constructeur prend comme paramètre une référence du plateau, la coordonnée x et la coordonnée y de l'emplacement de la case sur la grille. Une case avec un Flottant prend les mêmes paramètres en plus d'une référence sur un objet de type Flottant. Une *Case* avec un flottant a également un MouseListener qui sert à la

sélection avec le clique gauche de la souris. La case teste le type de son attribut flot pour lancer l'appel d'affichage approprié selon si c'est un naufragé, un bateau ou une proue. La classe a un attribut de type Color qui est une couleur gris-argenté pour le carré de sélection.

La classe *Bateau* a comme attribut une direction, un naufragé et une référence vers une case contenant sa proue. La classe instantie sa propre case et sa propre proue. *Proue* n'est que la tête du bateau. Selon la direction son orientation change.

2 Structures de données employées et justification

Comme structure de donnée, le programme utilise des *listes*. Contrairement aux tableaux, les listes permettent d'ajouter et de retirer des éléments pendant l'exécution du programme tout en mettant à notre disposition diverses méthodes pour manipuler son contenu. Nous utilisons cette structure pour enregistrer les joueurs de la partie, les paquets de cartes sur la table de jeu et pour chaque joueur, les paquets des adversaires.

Une liste de paquets est idéal pour les cartes sur la table, car leur nombre varie constamment au fil du jeu. Le choix d'une liste pour les joueurs est un choix personnel, un tableau aurait suffi car le nombre de joueurs ne varie pas pendant la partie. Chaque joueur a une liste d'adversaire qui est utilisé pour obtenir de l'information sur eux.

3 Algorithmes et justifications

Aucun algorithme n'a été utilisé pour la réalisation du programme. Son fonctionnement simple ne nécessitait pas l'implémentation de méthodes complexes ou même de l'utilisation de calcul mathématique.

4 Discussion

La programmation du projet était assez simple. Nous avons créé une classe *Joueur* pour regrouper les fonctionnalités concernant les joueurs, comme déterminer quel paquet voler, l'augmentation de son pointage, etc. Le fonctionnement de la partie de jeu est établis par la classe *Jeu* qui initialise les joueurs, le paquet de carte initiale, enregistre le nombre de partie et s'assure du bon déroulement. Pour structurer le tout, nos deux classes utilise la méthode *Update* qui est lancé à chaque tour pour les joueurs et qui exécute les méthodes de logiques pour leur actions du tour. La méthode *Update* dans la classe *Jeu* parcourt la liste des joueurs pour lancer leur méthode *Update* respective et elle en général, elle gère la déroulement logique de la partie. Cette structure nous permet de facilement rajouter plus de code logique si nécessaire dans le cas où nous voudrions un jeu plus avancé. Nous avons programmé de façon à ce que le nombre de joueurs, le nombre de paquets de cartes sur la table au début de la partie et le nombre de cartes dans

la main des joueurs peuvent être facilement modifiable à l'aide de constantes. Lorsque les joueurs performant des actions, un message décrivant ce qui se passe est imprimé dans la console. Le tout est facilité par l'implémentation de `toString()` dans la classe *Carte* pour afficher la couleur et la valeur numérique d'une carte. La partie de jeu imprime également plusieurs messages informant l'utilisateur de l'état du jeu. Il est donc facile de vérifier le bon déroulement de la partie à partir de la console.

5 Améliorations possibles

Malgré la possibilité de changer le nombre de joueur qui joue pendant une partie de paquet voleur, certaines caractéristiques ne sont pas compatibles. Par exemple, les joueurs n'ont pas d'identifiant unique pour identifier les actions à un joueur bien précis. La méthode pour déterminer le gagnant à la fin d'une partie ne peut que différencier que deux joueurs s'il y a une égalité dans le nombre de cartes contenue dans leur paquet. Il serait également envisageable de développer une interface graphique séparé du code logique du jeu pour pouvoir voir ce qui se passe. Comme dernier point d'amélioration, il serait bien de pouvoir laisser l'utilisateur jouer plutôt qu'exécuter des simulations sans interactions.