Rapport de projet

Ian RIVIERE

Table des matières

1 Introduction

Notre projet réalisé par Ophélie MARECHAL, Ian RIVIERE et Alexandre MARINE porte sur la réalisation d'un jeu de stratégie en tour par tour contre l'ordinateur dans le but d'appliquer les diverses techniques de POO ammenées à notre connaissance. Le joueur aura accès à un menu d'édition pour créer des champs de batailles ou cartes dans lesquels il pourra diriger ses unités en ayant pour but l'éradication totale des forces de l'ordinateur . Il pourra également sauvegarder et charger des parties et niveaux particuliers.

Dans un premier temps, nous présenterons la structure du projet à l'aide d'un diagramme UML. Ensuite nous verrons les différentes fonctionnalités présentes dans le jeu ainsi que les choix ergonomiques faits. Troisièmement nous présenterons la stratégie de jeu de l'ordinateur. Pour finir, une conclusion sur l'ensemble du travail effectué et du produit final concluera ce rapport.

Une bibliographie des ressources utilisées se trouve à la fin du rapport.

2 Structure du projet

Nous nous contenterons d'évoquer simplement les relations entre les classes principales du projet dans la partie écrite de cette section. Vous pourrez trouver un diagramme UML détaillé dans un fichier joint à ce rapport.

Le répertoire `images` est décomposé en sous répertoires pour une classification efficiente des ressources utilisées.

La méthode `main` se situe dans la classe `FenetreJeu` et va instancier la quasi totalité des ressources nécessaires pour le jeu.

2.1 Interfaces

Le projet utilise deux Interfaces permettant de choisir différentes valeur de paramétrage :
`Config` qui contient entre autres les dimensions (variables) de la grille de jeu et le niveau de zoom par défaut sur cette dernière.
`ISoldat` qui affecte des identifiants aux diverses unités.

2.2 Affichage

Les classes gérant les principaux éléments visibles à l'écran sont les suivantes :

- `MiniMap` qui permet de gérer la MiniMap pour le mode création et Jeu.
- `Bouton` qui possède un nom explicite sur son rôle.

— `LabelInfoSoldat` qui permet de faire apparaître des informations sur l'unité actuellement survolée ou sélectionnée. Notamment une description sommaire, une image et des statistiques sommaires sur la force de l'unité.
— `FenetreExplications` qui fait apparaître une petite fenêtre faisant office de tutoriel lors du lancement de la première partie d'une session de jeu.
— `FenetreInformation` qui permet l'ouverture de petites Fenêtres simples affichant des messages rapides, notamment durant le tour de l'ordinateur.
— `FenetreSauvegarde` qui gère la fenêtre s'ouvrant lorsque l'utilisateur désire sauvegarder ou charger un fichier .ser se trouvant dans le répertoire Sauvegardes.
— `FenetreFinPartie` est la classe qui permet l'ouverture d'une fenêtre particulière indiquant la victoire ou la défaite d'un camp.
— `PanneauCommun` est une classe centrale du projet, mère de `PanneauCreation` et `PanneauJeu`. Elle permet de définir le panneau principal qui sera affiché à l'écran.
— `PanneauCreation` est le panneau utilisé lorsque le joueur est en mode édition pour créer un niveau. Il permet de sélectionner divers Boutons permettant de façonner la carte hexagonale à la guise de l'utilisateur.
— `PanneauJeu` est le panneau utilisé lorsque le joueur est en jeu contre l'ordinateur. L'affichage de la grille de jeu utilise alors le système de Brouillard de Guerre et les options d'édition ne sont plus visibles. On peut alors jouer notre tour et y mettre fin grâce au bouton "Terminer le tour". Ce Panneau gère également les différentes phases et mécaniques de jeu.

2.3 Grille de Jeu

Notre classe `Grille` représente la grille de jeu hexagonale utilisée.

Cette contient un tableau de cases à deux dimensions, chase case étant un objet de la classe éponyme. Elle contient diverses méthodes : voisins d'une case, changement de type d'une case, recherche de chemin entre deux cases, calcul de distance, vérification si un obstacle est présent sur la trajectoire, calcul des cases visibles par les deux camps...

Ces méthodes sont utilisées principalement dans `PanneauCreation` et `PanneauJeu` pour manipuler la grille.

Une `Case` représente un emplacement dans la grille. Elle possède un coût de déplacement lorsqu'une unité essaie de rentrer dans cette dernière qui dépend de son type entier qui indique son contenu (herbe, eau, rocher, arbre...). Une case peut être traversable pour une unité (ce qui signifie qu'une unité peut se tenir sur cette dernière) et traversable pour un projectile (ce qui

signifie que si la trajectoire d'une attaque de portée supérieure à 1 passe par cette case, alors l'attaque fonctionnera ou non selon la valeur de retour de la méthode `pTraversable()`.

2.4 Soldats

Les classes `Heros` et `Monstre` héritent de la classe `Soldat` qui définie les méthodes et variables décrivant les statistiques d'une unité :
`HEAL,PV,PO,PM,PD,PVmax,PMmax,nbAT, nbATmax, nbHEAL,nbHEALmax`
nbAT et nbHEAL peuvent surprendre mais notre implémentation supporte des unités pouvant attaquer plusieurs fois par tour, soigner et attaquer en un tour etc ...

Un Soldat possède en plus de ces statistiques la connaissance de la case sur laquelle il se situe. Les méthodes utilisées pour les tours (getteurs et setteurs exclus) sont notamment `attaque()`, `soigne()`, `aJoue()`, `peutJouer()`, `newTour()` et `repos()`. Repos n'étant appelé que si une unité n'as rien fait pendant un tour ($\Leftrightarrow$ `!aJoue()`).

La classe `EnsembleSoldat` permet de gérer des ensembles d'unités pour garder trace des unités actuellement présentes sur la grille. Cette classe est notamment utilisé pour le tour des ennemis et la vérification des conditions de fin de partie (si l'un des deux ensembles parmi celui qui contient les Monstres et celui qui contient les Heros est vide, alors la partie est finie et le camp approprié remporte la partie).

3 Fonctionnalités

Le Joueur a accès a un menu de créations de niveaux lui permettant de préparer comme il le souhaite le champ de bataille avant de jouer en changeant la taille de la grille, les cases qui la composent, la répartition des unités sur le champ de bataille etc... Il peut affecter plusieurs cases pour la même modification rapidement en maintenant le clic droit enfoncé. Il est également possible de déplacer la carte en maintenant le clic gauche de la souris et en faisant glisser. Le zoom se fait avec la molette de la souris.
Il peut donc créer un niveau puis le sauvegarder pour le jouer plus tard en le chargeant. Les sauvegardes ont besoin d'un réperoitre nommé Sauvegardes dans le répertoire de lancement du programme.

Chaque camp possède 4 unités différentes possédant chacunes leurs spécifités en terme de capacités défensives, offensives et utilitaire. La vue de la carte est directement affectée par le positionnement des unités et leurs statistiques de portée et de déplacement. Les éventuels obstacles bloquent également la vue du joueur et des ennemis. Une unité peut être déplacée en plusieurs fois.

Si le joueur dispose son curseur sur un allié, il pourra voir comme l'indique la fenêtre d'explications la portée de déplacment de son unité et les cibles qu'elle peut soigner/attaquer. Si le joueur tente d'interagir avec une case qui n'est pas immédiatement à portée, un déplacement automatique aura lieu (si possible) permettant l'unité de mener à bien son action. Ainsi il n'est pas nécessaire de faire de nombreux clics redondants pour se déplacer et soigner/attaquer une unité.
Si le curseur est disposé sur une unité ennemie, alors seule sa portée de déplacement sera affichée pour que le joueur puisse visualiser les mouvements potentiels des unités ennemies.

4 Stratégie de l'ordinateur

Dans notre jeu, l'ordinateur n'est pas omniscient et dispose lui aussi d'un champ de vision limité par le positionnement de ses troupes. Lors de son tour, chaque unité va d'abord regarder les cases situés à une distance inférieure ou égale à la somme de ses PM(Points de Mouvement) et PO (Point de Portée) pour voir si il s'y trouve une cible d'intérêt : un ennemi à attaquer, un allié à soigner... Si oui alors il effectuera l'action appropriée. Avant de se préparer à se déplacer vers la prochaine cible d'intérêt si il y en a une.

Si aucune cible n'est à portée immédiate, alors l'ennemi se déplacera vers la cible d'intérêt la plus proche de lui dans le champ de vision de toute l'armée des Monstres. Sinon l'armée des monstres ne voit aucune cible d'intérêt, l'unité en profitera pour se reposer et regénérer des PV (Points de vie).

5 Conclusion

5.1 Répartition du travail

Ophélie MARECHAL a réalisé la totalité des ressoures graphiques (images et sprites) utilisées dans ce projet et la majorité des fonctions d'affichage et fenêtres. Ian RIVIERE a codé les fonctions de distance, pathfinding, visibilité, le brouillard de guerre, l'IA des ennemis et la quasi totalité des mécaniques de jeu . Alexandre MARINE a codé la génération aléatoire de cartes, les Soldats et leur équilibrage. Il a également réalisé le diagramme UML. Chaque membre du groupe a codé la javadoc au fur et à mesure de l'avancement.

5.2 Conclusions diverses

Dans l'ensemble nous trouvons notre projet plutôt complet et réussi malgré l'absence de sons notamments et d'animations plus poussées. Les exceptions ont peu été utilisées dans notre projet car les types de retour de nos fonctions et leur fonctionnement ont déjà été pensé pour des tests rigoureux

sur le bon fonctionnement du jeu.

Ressources utilisées :

1. https ://www.redblobgames.com/grids/hexagons/
2. TPs et cours de l'année
3. Ressources graphiques par Ophélie MARECHAL