Université de Mons
Faculté des sciences
Département d'Informatique

Projet d'informatique : Quoridor

Rapport de projet

Professeur :
Hadrien Melot

Auteurs :
Arnaud Palgen
Fabio Cumbo

Année académique 2015-2016

Table des matières

1 Introduction

Dans le cadre du Projet d'informatique, nous avons dû réaliser le jeu du Quoridor par groupe de deux. Dans ce rapport sera détaillé les principaux choix effectués, les points forts et les points faibles de notre programme, les apports positifs et négatifs du projet, les éventuels bugs de notre programme et un petit guide utilisateur qui explique comment utiliser notre programme.

2 Répartition des tâches

Notre choix s'est porté sur la division du travail en deux parties : L'interface graphique et la logique. Fabio a réalisé l'interface graphique sauf la représentation du tableau de jeu qui, elle, a été réalisée par Arnaud qui a également réalisé la logique.

3 Choix effectués

3.1 Tableau de jeu

Construction du tableau

Deux méthodes on été envisagées :

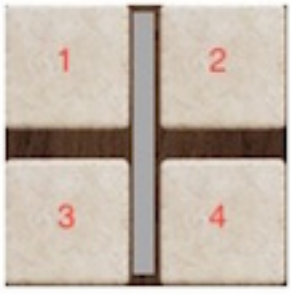
- Créer un tableau à deux dimensions de taille 9x9 contenant des objets cases (classe Box) où chaque objet possède des boolean (ex : isMur_d est true s'il y a un mur à droite de la case) pour savoir s'il y a un mur en haut, en bas, à gauche ou à droite d'une case.
- Créer un tableau à deux dimensions de taille 17x17 contenant des objets cases (classe Box) et des objets murs (class Wall).

La première méthode a été choisie. Son avantage, comparé à la deuxième, est que, par exemple, pour savoir s'il y a un mur en haut d'une case, l'information est directement disponible sur un objet. Il n'est pas nécessaire de devoir accéder à l'élément à la ligne du dessus.

Par la suite, des boolean ont été ajoutés pour savoir si la case est sur un bord.(ex : isCanMur_d est false si la case est à une extrémité droite du tableau de jeu) et les boolean isMur_ ont été remplacé par des références vers un mur pour simplifier la méthode qui vérifiant le placement ou non d'un mur (canPutWall()).

Placement des murs

Pour situer un mur, on a choisi de créer un tableau à deux dimensions contenant les cases entre lesquelles le mur passe.



Par exemple dans l'image ci-dessus, un mur vertical est placé entre les cases 1, 2 et les cases 3, 4. Le tableau à deux dimensions contiendra alors {{1, 2},{3, 4}}.

Déplacement des pions

Pour le déplacement des pions, deux méthodes ont d'abord été créées : une méthode *canMove-Piece(BoxmaCase)* qui retourne true si un pion peut bouger sur la case *maCase* et une méthode qui effectue le mouvement.
Par la suite, pour des raisons de facilités, et surtout pour le pathFinder et donc les intelligences artificielles, la méthode canMovePiece(Box maCase) à été modifiée en une méthode *whereMove-Piece(Box maCase)* qui retourne un tableau de cases sur lesquelles le joueur peut se déplacer à partir de la case *maCase*.

3.2 PathFinder

Après avoir pris connaissance de plusieurs algorithmes de recherche de chemin comme **a*** et **Dijkstra** via des recherches sur internet, nous avons décidé d'en réaliser un nous-même et avons opté pour la récursivité. Un premier algorithme a été écrit mais n'ayant pas fait attention à ce que nous avions vu en programmation et Algorithmique 1, il était incorrect. Un deuxième Pathfinder récursif a ensuite été écrit. Il était fonctionnel mais pouvait donner un chemin trop long, ce qui n'était pas idéal pour l'intelligence artificielle facile. Ensuite, Pierre Zielinski nous a aiguillé sur le principe d'un pathFinder qui donne le plus court chemin entre une case de départ et une case d'arrivée. Le principe de ce pathFinder est qu'il avance simultanément dans toutes les directions. Dès qu'il arrive sur une case d'arrivée, il s'arrête et renvoie toutes les cases parent de la case d'arrivée. Comme il avance simultanément dans toutes les directions et qu'il s'arrête dès qu'il est sur une case d'arrivée, le chemin qu'il retourne est le plus court.

Pour le pathFinder, un nouveau tableau à deux dimensions contenant des *PathCases* à été créé pour faciliter l'utilisation du PathFinder. C'est à dire qu'après chaque chemin cherché, le tableau du pathFinder est réinitialisé. Le fait d'avoir deux tableaux distincts, permet de ne pas risquer de modifier le tableau de jeu logique.

3.3 Intelligences artificielles

Deux intelligence artificielles ont été intégrées au projet :

La première qui est "aléatoire", c'est-à-dire qu'elle choisit aléatoirement de poser un mur ou de bouger son pion. Pour bouger son pion, elle choisit une case aléatoirement dans la liste des cases où elle peut aller. Pour placer un mur, elle choisit les coordonnées d'une case aléatoirement (compris entre 0 et 9 qui est la longueur du tableau) et les autres cases sont choisies pour que cela correspond à un ordre de case correcte de placement de mur..

La seconde qui est "facile", récupère via le pathfinder son chemin et le chemin de son adversaire. Si son chemin est plus court que celui de son adversaire, elle bouge son pion sur la case suivante de son chemin, sinon, elle met un mur sur le chemin de l'adversaire. Pour ce faire, pour chaque case du chemin de l'adversaire, elle regarde où se trouve la prochaine case, si la suivante est en haut de l'actuelle, elle va poser un mur en haut de la case actuelle, idem pour les autres directions.

3.4 Interface graphique

Pour la représentation graphique du plateau de jeu, la méthode choisie a été de créer un tableau de 19X19 construit sur base du tableau logique et contenant des objets Contents (Contents est une classe abstraite qui a comme classe enfant imgWall, imgBox et imgIntersection). Le but de ce tableau est de le parcourir plus facilement que le tableau logique pour en dessiner sa représentation à l'écran. L'interêt de cette représentation porte notamment sur l'aspect esthétique du plateau de jeu. Tous les élements du plateau sont donc représentés par une image et donc nous n'avons pas eu

besoin d'utiliser un layout pour les placer. Par conséquent nous avons du gérer le déplacement des pions et le placement des murs grâce à l'implémentation de l'interface MouseListener
Pour améliorer l'ergonomie, nous avons également du créer un menu pour pouvoir choisir le mode de jeu voulu (mode 1 joueur, Ia vs Ia, 2 joueurs) séléctionnable à l'aide de boutons personnalisés.(classe Button héritant de JButton) Les boutons du menu sont placés à l'aide d'un GridBagLayout.

Afin d'éviter toute incompréhension avec l'utilisateur, nous avons inséré un JPanel(sélectionnable à partir du menu) qui s'occupe d'afficher un JLabel(en langage html) expliquant les règles de jeu.

4 Points forts du projet

L'aspect esthétique est jolie. Facile à utiliser, tout le monde peut y jouer sans jamais connaitre le jeu.

5 Points faibles du projet

-Pas de mode de jeu supplémentaire -

6 Erreurs connues du programme

Une erreur au niveau de la sauvegarde

7 Apports positifs et négatifs du projet

7.1 Apports négatifs du projet

La réalisation du projet a pri plus de temps que prévu initialement. La principale difficulté est d'apprendre le langage Java et réaliser le projet simultanément.

7.2 Apports positifs du projet

Devoir réaliser un projet de taille conséquente, nous a appris à découper un programme, travailler en équipe, s'organiser et à respecter des délais.

8 Guide utilisateur

Menu d'accueil

A partir du Menu d'accueil, vous avez quatre possibilités :

- Cliquer sur *2 joueurs*, pour démarrer une partie de deux Joueurs (humains)
- Cliquer sur *1 joueur*, pour jouer contre l'ordinateur. Vous devrez ensuite choisir un niveau de difficulté : normal ou facile.
- Cliquer sur *Ordi. VS Ordi.* pour lancer une partie où l'ordinateur jouera contre lui même. Vous pourrez ensuite choisir les niveaux d'intelligences artificielles qui s'affrontent.
- Cliquer sur *règles du jeu*, pour lire les règles du jeu.

Déplacer son pion

Pour déplacer son pion, cliquez sur la case sur laquelle le pion doit se diriger. Les cases possibles seront marquées par un cercle gris.

Poser un Mur

Pour mettre un mur, tracez le mur là où vous voulez le placer. Les murs ont une longueur de 2 cases.

Sauvegarder une partie en cours

Pour sauvegarder une partie, cliquez en haut à droite sur *fichiers*, puis *sauvegarder*. Choisissez ensuite le nom du fichier de sauvegarde et le répertoire où celui-ci doit être sauvé.

Reprendre une partie

Pour reprendre une partie, cliquez sur *fichiers* puis sur *importer*. Choisissez ensuite le fichier de sauvegarde à partir duquel vous voulez reprendre la partie.

Utiliser Ant

- *ant build* : pour compiler toutes les classes du projet
- *ant clean* : pour supprimer le répertoire bin qui contient tout les fichiers .class
- *ant run* : pour exécuter le jeu
- *ant stat -DnbrPartie=x -DTypePlayer1=x DTypePlayer2=x* : pour lancer la classe de statistiques. L'argument *nbrPartie* est le nombre de parties que la classes MainStat doit faire joueur . Les arguments *TypePlayer1* et *TypePlayer2* sont les type tu premier et du deuxième joueur : soit *aleatoire*, soit *facile*.
- *ant test* : pour executer les tests JUnit.

9 Conclusion

Ce projet nous a permis principalement de prendre de l'experience dans ce domaine et d'approfondir nos connaisances sur ce langage et de la programmation en général.

10 Annexes