TP 2

Faculté des sciences et technique de Tanger
Cycle ingénieur logiciel et systèmes informatiques
2015/2016

Compte rendu TP2

Intelligence artificielle

ENCADRE PAR :

- HANAFI HAMZA
- TABASSAMET HAMZA

ENCADRE PAR :

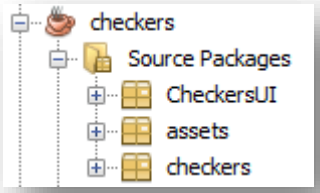
- M'HAMED AIT KBIR

Table des matières

Introduction

On cherche à mettre au point une application en JAVA permettant de simuler le jeu des dames aussi connu sous le nom Checkers en anglais, pour implémenter les stratégies de recherche adversiale vus en cours, notamment Minimax à profondeur limitée, avec ou sans prise en compte de l'élagage $\alpha - \beta$

Pour ce faire, nous avons adapté le code du projet du jeu Tic-Tac-Toe, la figure suivante représente l'ensemble des packages de notre projet.

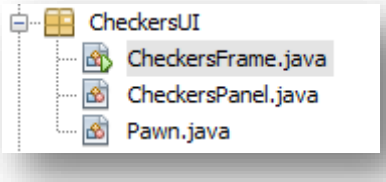


Notre application comprend 3 packages, dont chacun contient un ensemble de classes ou bien des ressources (images, musique etc.), l'organisation et la hiérarchie de nos packages est la suivante :

Un package (CheckersUI) contenant les classes permettant l'affichage d'une interface graphique convenable pour le jeu, un deuxième nommé (assets) qui englobe l'ensemble des images utilisées et la musique, finalement le package (checkers) qui contient les classes et la logique permettant de simuler le jeu des dames.

Réalisation

Package CheckersUI :



La classe principale du package est la classe CheckersFrame elle hérite de la classe JFrame et implémente quelques écouteurs *(Listeners)*, elle permet de lancer le jeu et donne la main au joueur.

Elle permet ainsi de créer un menu qui permet de changer les paramètres du jeu comme par exemple choisir la difficulté du jeu, changer le style du damier, changer l'algorithme du jeu,

et bien encore des boutons pour revenir en arrière ou en avant, demander de l'aide, et réinitialiser le damier.

Effectuer un mouvement :

Voici les étapes pour effectuer un mouvement :

1. Le joueur peut déplacer graphiquement son pion en le glissant vers la position souhaitée avec sa souris.
2. Lorsqu'il relâche la méthode *mouseReleased* est exécutée pour récupérer la nouvelle position du pion.
3. Exécution de la méthode *moveUser* pour déplacer le pion.

La méthode *moveUser* prend en paramètre la position initiale du pion et sa nouvelle position et procède comme suit :

1. Vérifier si le mouvement effectué est autorisé.
2. Vérifier si le mouvement est un saut sur un autre pion.
3. Exécution de la méthode *executeUserJump* si le point 2 est vérifié.
4. Vérifier si un mouvement saut existe.
5. Un message disant : « Mouvement non valide. Si vous pouvez sauter, vous devez. » est affiché si le point 4 est vérifié.
6. Vérifier si d'autres sauts sont possibles.
7. Le joueur doit compléter son mouvement si le point 6 est vérifié.
8. Exécution de la méthode *executeMove* si les points 2 et 4 n'ont pas été vérifiés.

La classe *CheckersPanel* hérite de la classe *JPanel*, et elle s'occupe de l'affichage des éléments du jeu : le damier, les pièces.

Donner de l'aide :

Deux façons pour donner de l'aide au joueur :

1. En cliquant sur le help dans le menu options en rouge : cela permet de montrer au joueur de quelle pièce doit-il jouer et où doit-il la mettre.
2. En cliquant sur une pièce : cela permet de montrer au joueur où peut-il jouer avec cette pièce.

Voici les étapes pour répondre et une demande d'aide :

1. Exécution de la méthode *findAllValidMoves* pour trouver tous les mouvements et sauts possible, ces mouvements sont renvoyés dans une liste.
2. Dans le cas d'un clic sur un pion on cherche dans la liste les mouvements possibles pour le pion.
3. Dans le cas d'un clic sur le bouton *HELP* on cherche le meilleur mouvement à l'aide de la méthode *bestmovesfromhelp*.

La classe *Pawn* représente les pions du damier, et permettre la gestion des pions.

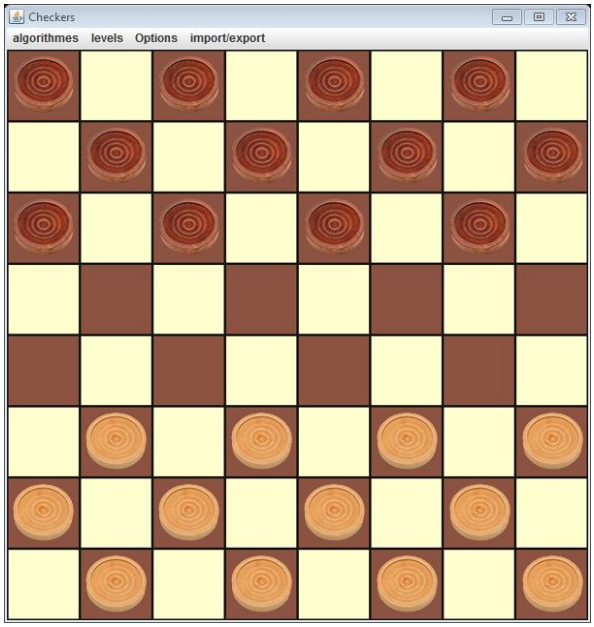
Changer la difficulté du jeu :

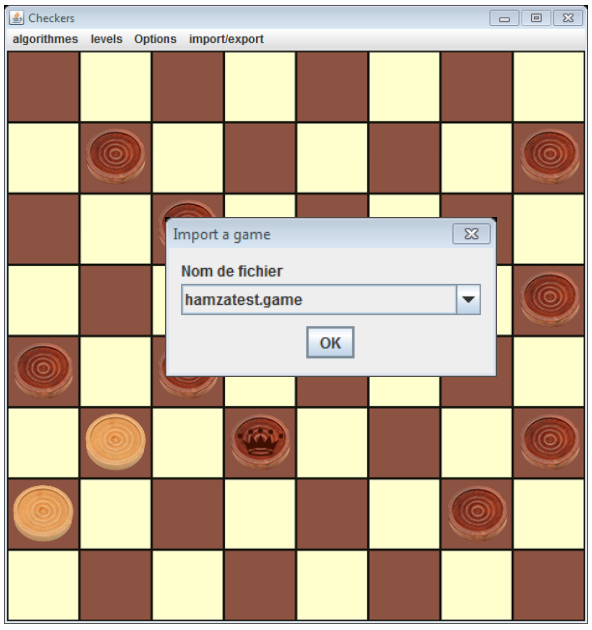
Il s'agit ici de modifier la profondeur de l'algorithme, pour cela nous avons affecté aux trois niveaux (facile, moyen, et difficile) du jeu les profondeurs suivantes :

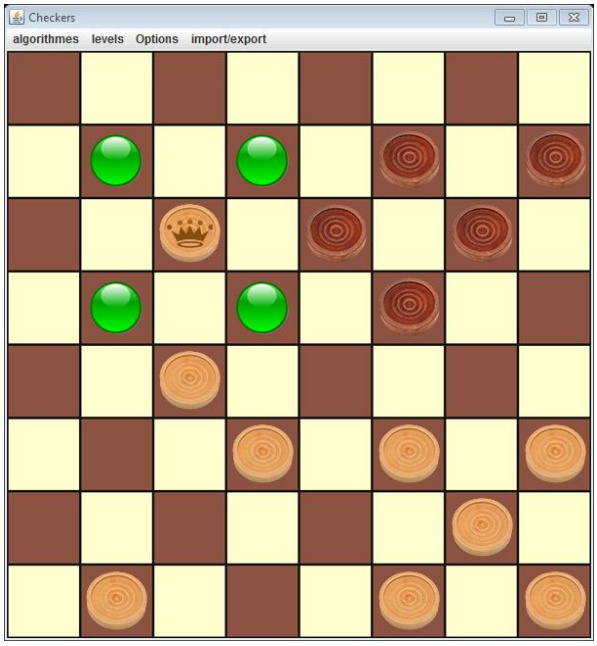
- Facile (EASY) : 2
- Moyen (MEDIUM) : 5
- Difficile (HARD) : 8

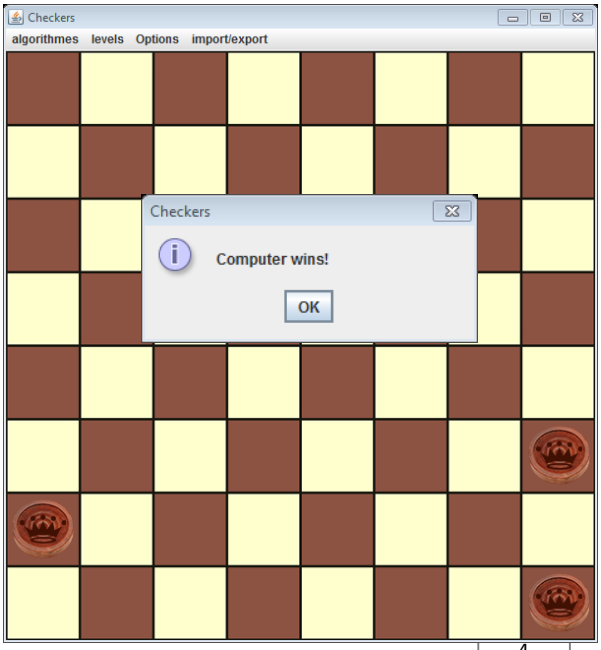
Interface graphique du jeu :

L'interface graphique du jeu est simple et facile à manipuler à chaque instant pendant le jeu se déroule ou en début de partie, voici quelques captures d'écrans :

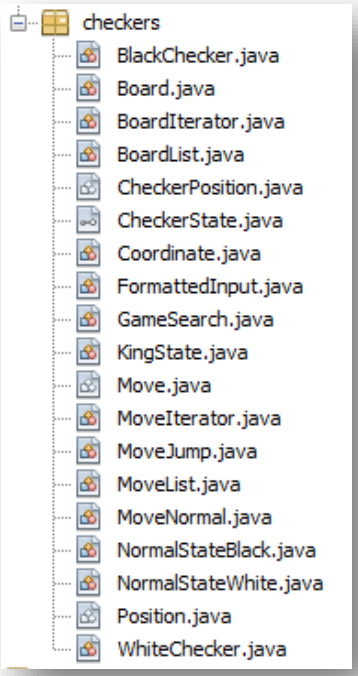








Package checkers :



Les classes et leurs rôles :

- *Board :* cette classe représente le damier, elle contient des méthodes permettant d'initialiser le damier et de l'afficher en consol ainsi l'enregistrement des coûts joués.
- *BoardIterator :* c'est une classe permettant de définir un *iterator* pour la classe *BoardList.*
- *BoardList :* représente une liste des damiers et permet de créer un lien entre ces derniers.
- *BlackShecker, WhiteShecker* : ces deux classes héritent de la classe CheckerPosition et permettent de gérer les pièces de chacun des joueurs.
- *CheckerPosition :* cette classe hérite de la classe Position et permet de définir une position dans le damier.
- *CheckerState :* cette une interface *(State pattern*) pour les méthodes : *findValidMoves* et *findValidJumps*, elle permet de trouver les mouvements et les sauts possibles selon l'état de la pièce (pièce normale ou pièce dame).
- *Coordinate :* c'est une classe qui représente les coordonnées des cases dans le damier, et permet aussi de se déplacer dans le damier.
- *KingState :* définit l'état d'une pièce dame et permet de trouver les mouvements et sauts possibles pour ce type de pièce.
- *Move :* c'est une classe abstraite permet de définir un mouvement.
- *MoveIterator* : c'est une classe permettant de définir un iterator pour la classe MoveList.
- *MoveList* : représente une liste des mouvements et permet de créer un lien entre ces derniers.

- *MoveJump :* c'est une classe permettant de définir les mouvements sauts, elle hérite de la classe *Move*.
- *MoveNormal :* c'est une classe permettant de définir les mouvements normaux, elle hérite de la classe *Move*.
- *NormalStateBlack* : c'est une méthode qui implémente l'interface CheckerState et définit les méthodes *findValidMoves* et *findValidJumps* de la class *CheckerPosition* pour les pièces noires.
- *NormalStateWhite :* c'est une méthode qui implémente l'interface *CheckerState* et définit les méthodes *findValidMoves* et *findValidJumps* de la classe *CheckerPosition* pour les pièces blancs.
- *GameSearch :* cette classe permet de définir les règles du jeu des dames, elle définit ainsi les algorithmes de recherches : *minimax* avec une profondeur limitée, et *minimaxAB* avec une profondeur limitée sans prise en compte de l'élagage $\alpha - \beta$.

Conclusion

Ce TP nous a permet d'approfondir nos connaissance en programmation orienté objet JAVA, ainsi d'appliquer les algorithmes de recherche vus en classe à savoir Minimax à profondeur limitée, avec ou sans prise en compte de l'élagage $\alpha - \beta$, et de voir la différence entre les deux versions de l'algorithme en terme de vitesse.