Projet Chess

Pierre Jobic - Rémi Dupré

Février 2017

1 Introduction

Le but du projet était de programmer un jeu d'échecs et une IA qui joue de manière complètement aléatoire.

2 Apperçu du code

2.1 Présentation des différents fichiers du projet

2.1.1 Structure du projet

Il y a 6 fichiers scala dans notre projet :
— Game : une partie d'échec, juste la gestion des règles
— Piece : représentation et calculs sur une pièce
— Interface : l'affichage, et le choix des mouvements
— Joueur : le comportement des deux types de joueurs
— Menu : pour choisir le type de partie
— Main : juste là par habitude

2.1.2 Game

La classe Game représente le déroulement général d'une partie. Et l'interface pour demander les mouvements. Ne s'encombre pas du comportement des joueurs ou de l'affichage de la partie

Représentation du plateau Le plateau est représenté par une liste des pièces, dans un ordre arbitraire. On a géré la condition de victoire et de pat en appelant la fonction "every_possible_move" qui renvoie la liste des mouvements possibles pour un joueur (et regarde si cette liste est vide).

2.1.3 Interface

L'interface est minimaliste, composée d'une simple grille de boutons et ne prend en compte que deux interactions. Elle s'occupe de stocker l'avancement dans le tour d'un joueur humain (s'il a sélectionné une pièce, attend que le joueur dise où déplacer la pièce.

2.1.4 Joueur

Pour l'encapsulation, le choix a été d'interfacer le joueur en lui donnant accès à l'objet "game" et en définissant une méthode "wait_move" qui attend que le joueur face son déplacement dans la partie. Intégrer l'IA aléatoire a été extrêmement rapide et simple, toutes les fonctions nécessaires étaient déjà présentes.

2.1.5 Pièce

Représente une pièce en en stockant la position, la couleur et le rôle. Elle a accès à la partie dans laquelle elle est implantée, ce qui permet de définir une fonction retournant l'ensemble des positions auxquelles elle peut accéder.

3 Répartition

Assez nous n'avions pas défini de découpe précise au début du projet, cependant une tendance a finalement été très nette :

— Pierre s'est occupé de la gestion des règles, des déplacements, et du déroulement de la partie

— Rémi s'est plutôt occupé de la structure générale et de l'interface.

Cette tendance était finalement quasiment respectée mais pas systématique. Nous n'avons pas rencontré de difficulté majeure à travailler simultanément, nos codes se sont même emboîtés sans trop de difficultés.

4 Remarques post-développement / Conclusions

— Pour ce qui est des problèmes dans notre programme, nous nous somme assez vite dit que la structure de liste n'était pas très utile pour représenter le jeu. Et bien qu'elle n'entraîne pas de contrainte majeure, n'apporte aucune clarté supplémentaire par rapport à l'utilisation d'un tableau, mais seulement un léger sur-coût à la recherche d'une pièce en case (i, j).

— Nous sommes resté sur un résultat assez minimaliste, mais nous aurions bien aimé rajouter quelque features, par exemple proposer une IA plus intéressante.

— Le fait que n'ayons pas eu de problème à combiner nos codes semble assez positif quand à la souplesse de celui-ci, et en tous cas il semble suffisamment clair pour pouvoir être étendu.

— Du côté de l'interface, il reste quelque-chose d'un peu sale que je (Rémi) n'ai pas cherché à résoudre : chaque tour de l'IA est lancé dans un nouveau thread, ce qui masque un problème antérieure de l'interface qui ne se métais pas à jours après un tour de l'IA.