Projet chess Part 2

Rémi Dupré et Pierre Jobic

March 2017

1 Introduction

Le but de cette partie était de compléter le jeu d'echec de la première partie. La première partie ne gérait que les mouvements les plus simples des pièces et n'avait pas de système de sauvegarde, ni de la prise en compte d'un timer, ni de gestion du pat. Nous avons donc implémenté ces derniers points et programmé une variante du jeu d'echec : Proteus.

2 Apperçu du code

Nous allons voir les ajouts et les changements importants dans le code de notre projet.

2.1 Remarques générales

Nous avons changé les types de donnés (on l'a fait directement après vos remarques sur notre projet). Maintenant notre board est une matrice 8x8 (alors qu'avant c'était une liste de pièces). Ce qui a donc engendré beaucoup de changement dans notre code.

Nous avons pas spécialement fait des efforts sur la complexité de nos programmes étant donné que nous n'avons pas vraiment rencontré des problèmes de delai ou autre lorsqu'on faisait des IA vs IA en partie rapide.

Comme dans la première partie du projet, Rémi a plutôt géré l'interface, le timer, la gestion de la game tandis que Pierre a plutôt géré les nouveaux mouvements des pièces, et les pat. Le système de sauvegarde et la variante ont été fait à deux.

2.2 Variante du jeu d'échec

(règle à partir de http://www.sjgames.com/proteus/) Proteus est la variante du jeu d'échec que nous avons choisi car la programmation orientée objet de scala nous permettait facilement de programmer la nouvelle variante. Le point le plus compliqué était de gérer la class Dice (de Proteus) car les pièces peuvent changer de class au cours de la partie.

2.3 Structure du projet

Il y a 10 fichiers scala dans notre projet (dont 3 nouveaux):

- 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
- (new) savesystem : gestion de la sauvegarde (format PGN)
- (new) Proteus : variante de notre jeu d'échec
- (new) Timer : Une gestion du rythme de jeu imposé
- (new) tools : fonctions utiles qui n'ont pas leur place ailleurs.

2.3.1 Game

Nous avons principalement ajouté/modifié la fonction "move" (qui gère tous les mouvements des pièces au cours d'une partie, et la gestion des pat). Le fait d'avoir un système de sauvegarde a été très utile pour programmer les pat (car il faut souvent revenir en arrière dans la partie pour la triple répétition d'une position ou la règle des 50 coups).

2.3.2 Pièce

Nous avons dû différencier l'endroit où les pièces menaçaient des cases, et l'endroit où les pièces peuvent se déplacer ! (car par exemple le roque n'est pas un mouvement d'attaque) Nous avons aussi ajouté une fonction $move_to$ pour la classe abstraite pièce qui permet de bouger les pièces comme il convient (cette fonction est surtout là pour alléger l'implémentation de move qui est dans Game et aussi pour gérer le changement de position de la tour lors d'un roque, et de la suppression d'un pion lors d'une prise en passant)

2.3.3 save system

Nous avons voulu gérer la sauvegarde comme un arbre. C'est à dire que d'une position de la partie (pour commenter une partie par exemple) il peut être intéressant d'explorer les différentes possibilités à partir de cette position. Ce qui donc une structure d'arbre. (C'est pourquoi save est définit de manière recursive avec une List[save]). En pratique pour charger une partie, on prends toujours le premier élément de la List[save]. Et pour la sauvegarde on ajoute la save en premier élément de la List[save].

La gestion de sauvegarde est assez couteuse en complexité (notamment pour les pat, nous avons donc quand même fait un peu attention à la complexité de nos fonctions même si lors d'un chargement d'une partie longue, il y a un temps d'attente)

Nous avons gérer un format PGN assez simple (sans commentaire) et de la forme "n. (A)a0 (A)a0" (donc sans tiret, ni de x quand il y a une prise)

2.3.4 Proteus

(règle à partir de http://www.sjgames.com/proteus/) la gestion de fin de partie est beaucoup plus simple ainsi que la gestion des échecs car le roi n'existe pas dans cette variante ce qui simplifie beaucoup de chose. (Notamment tous les mouvements interdits qui mets notre roi en échec) On a dû rajouté la classe *pyramid*, ainsi que $pawn_proteus$ dans pièce car la *pyramid* est une nouvelle pièce et que les pions dans proteus peuvent toujours faire un saut de 2, mais la prise en passant n'existe pas. Dans le cas général, les mouvements des pièces sont gérés en remplacant une pièce de protéus (un dé) par la pièce correspondante dans le jeu standart, d'extraire les mouvements possibles et de remettre la pièce d'origine ensuite.

3 Remarques post-développement / Conclusions

- Contrairement à la première partie du projet, on touchait un peu plus à des codes communs, nous avons donc dû s'organiser un peu plus.
- Comme à la première partie, nous avons un résultat un peu minimaliste. On espère que la troisième partie sera sur la programmation d'une IA.
- Je (Pierre) m'excuse auprès de Rémi pour ne pas pouvoir être présent vendredi lors de la soutenance.