JEU DE KHAN

P16

Rapport de projet – IA02

Rapport de projet IA02 – Jeu de Khan – Printemps 2016

Par Nicolas DEJON et Titouan GALOPIN

Introduction

Le projet « Jeu de Khan » de IA02 (Printemps 2016) a eu pour objectif de développer en Prolog une version sur ordinateur du jeu de plateau « Jeu de Khan » afin de mettre en pratique les connaissances de Prolog et d'intelligence artificielle découverts durant le semestre.

Il a aussi et surtout consisté en la réalisation d'une intelligence artificielle pour ce jeu permettant de jouer contre l'ordinateur et de laisser l'ordinateur jouer contre lui-même.

Etape 1 : initialisation du plateau et prédicats d'affichage

Choix de la structure de donnée du plateau

Le jeu de Khan est composé d'un plateau, chaque case ayant une valeur et pouvant accueillir une pièce. Nous avons fait le choix de réaliser une seule liste représentant à la fois le plateau, les valeurs des cases et les pièces en jeu, pour simplifier l'accès aux informations.

Ainsi, un plateau peut être représenté de la forme suivante :

```
[
    [3, [1,kao], 2 , 2, 3, 1],
    [[2,sr5], 3, [1,sr1], [3,sr2], [1,sr3], [2,sr4]],
    [[2,kar], [1,so1], [3,so2], [1,so3], [3,so4], [2,so5]],
    [1, 3, 2, 2, 1, 3],
    [3, 1, 3, 1, 3, 1],
    [2, 2, 1, 3, 2, 2]
]
```

Comme vous pouvez le voir, chaque case est caractérisée par :

- Ses coordonnées dans le tableau à deux dimensions
- Sa valeur
- Et sa pièce (s'il y en a une, dans ce cas la case est une liste)

L'intérêt de cette structure réside dans sa simplicité d'utilisation au sein d'algorithmes récursifs : comme toute l'information nécessaire pour travailler avec une case est stockée dans le même tableau, un seul parcours récursif suffit.

Initialisation

Le projet doit supporter trois types de jeu : Humain vs Humain, Humain vs Ordinateur et Ordinateur vs Ordinateur. Nous avons donc établi trois prédicats différents, selon le type de jeu souhaité :

- *initHumanVsHumanBoard*

Initialise le plateau de jeu en demandant aux deux joueurs quelles positions ils souhaitent avoir.

- *initComputerVsComputerBoard*

Choisis une position initial aléatoire parmi un subset des placements possibles pour les deux ordinateurs en respectant les règles de positionnement. Cette partie pourrait être améliorée en termes d'intelligence artificielle mais nous avons concentré notre travail sur l'intelligence artificielle en cours de jeu.

- *initHumanVsComputerBoard*

Fonctionne à la fois comme *initHumanVsHumanBoard* pour demander au joueur et comme *initComputervsComputerBoard* pour mettre en place un positionnement aléatoire pour l'ordinateur.

Affichage

Le prédicat *displayBoard(B)* affiche un plateau donné en paramètre de manière graphique.

Etape 2 : mouvements possibles et déplacement des pièces

Mouvements possibles

- *allowedMove(Player, Board, NewX, NewY, Piece)*

Vérifie que le déplacement donné est autorisé dans la situation actuelle, à partir du khan actuel et des règles du jeu.

- *possibleMoves(Player, Board, PossibleMoves)*

Renvoi la liste des déplacements possibles à partir d'une situation donnée, pour un joueur donné.

Déplacement des pièces

- *movePiece(Player, PieceMoved, X, Y, Board, NewBoard, OldPiece)*

Déplace une pièce donnée (PieceMoved) vers une nouvelle position (X, Y).

- *selectMove(Player, X, Y, Board, NewBoard, OldPiece)*

Demande un déplacement à l'utilisateur et l'effectue en appelant movePiece.

Etape 3 : intelligence artificielle

Idée initiale

Dans un premier temps, nous avions réfléchi à implémenter l'intelligence artificielle de manière assez simple mais pragmatique. En utilisant *possibleMoves*, nous avons la liste des coups possibles dans une situation donnée. En itérant sur cette liste de coups possibles, nous aurions pris le premier coup dans l'ordre suivant :

- Un coup nous permettant de gagner (manger la kalista adverse)
- Un coup nous évitant de perdre (notre kalista serait mangée)
- Un coup nous évitant de nous faire manger une pièce
- Un coup mangeant une pièce adverse

Si aucun coup n'avait été trouvé de cette manière, nous nous serions rabattus sur un coup au hasard.

Cette idée initiale, bien que simpliste, proposait un ordinateur suffisamment intelligent pour ne pas perdre quand il n'est pas obligé, et pour gagner quand il le peut. Cependant, grâce aux techniques découvertes en IA02, nous avons tenté d'aller plus loin, particulièrement avec l'utilisation de *minimax*.

Utilisation de minimax

L'algorithme de l'intelligence artificielle se base sur minimax. Les prédicats principaux sont :

- *generateMove(Player, X, Y, Board, NewBoard, OldPiece)*

Point d'entrée de l'algorithme de determination du meilleur coup dans une situation donnée. Ce prédicat appelle les autres prédicats de l'IA.

- *bestMove(Player, Board, OldN, NewN, [X,Y])*

Calcule un score stocké dans N évaluant une position. Ce prédicat fait appel à calculateMovesCost, qui lui fait appel à evaluateSituation. Comme spécifié par l'algorithme minimax, evaluateSituation fait appel à bestMove selon la position adverse. Le prédicat bestMove est donc indirectement récursif.

- *calculateMovesCost(Player, Board, [[[X,Y], Val, Piece]|Q], OldN, CostedPossibleMovesList)*

Prend en entrée la liste des coups possibles à un moment donné et calcul leur score en utilisant evaluateSituation.

- *evaluateSituation(Player, Piece, X, Y, Board, OldN, Khan, NewN)*

Calcule un score pour la situation actuelle. Ce score se base sur les règles suivantes :

  - Si le coup nous permet de prendre la kalista adverse, le score est -100
  - Sinon, le coup a une valeur de 10 par déplacement (choix du chemin le plus court)
  - Enfin, quand des coups ont la même valeur, un coup au hasard est choisi parmi eux

Selon le principe de minimax, chaque évaluation d'une situation consiste en l'évaluation de la situation potentielle adverse et la nôtre de telle sorte que nous maximisions le score de la situation adverse (position la moins favorable) tout en minimisant la nôtre (position la plus favorable).

Etape 4 : boucle de jeu

La boucle de jeu est constituée de 4 prédicats principaux :

- *main()*

Prédicat principal : il lance l'application en elle-même. Selon le type de jeu demandé, il appelle *start(Type)*.

- *start(Type)*

Démarrer un type de jeu (HvsH, HvsC, CvsC) demandé.

- *runHumanVsHuman / runHumanVsComputer / runComputerVsComputer*

Exécuter un tour pour le type de jeu associé.

- *victory(Player, OldPiece)*

Etablit si le joueur donné gagne la partie (permet d'arrêter la boucle).

Difficultés rencontrées et améliorations possibles

Il a été difficile de s'adapter à la logique de Prolog, qui est complètement différente de celle des autres langages de programmation. Le principal problème que nous avons rencontré durant le projet fut donc le temps, nécessaire pour être capables d'utiliser le langage correctement.

Nous avons eu quelques difficultés à mettre en place minimax, particulièrement au niveau de la récursivité sur trois prédicats (bestMove appelle calculateMovesCost qui appelle evaluateSituation qui appelle récusrivement bestMove). En effet, cette récursivité complexe, bien que puissante, est très difficile à débugger étant donné que lorsqu'un bug se produit, il se produit dans un état très difficile à reproduire.

Il est possible d'améliorer le projet à plusieurs niveaux, particulièrement :

- Au niveau de l'affichage, ou il serait possible d'utiliser des couleurs pour améliorer la lisibilité
- Au niveau de l'intelligence artificielle en déterminant une meilleure évaluation des coups (par exemple basée sur le fait de prendre des pièces ou non, ou encore sur le fait qu'une pièce bloquera un mouvement adverse gênant)
- Au niveau du code en lui-même, où il est possible d'améliorer la maintenabilité et la lisibilité en le reprenant et en le réorganisant. Nous avons découvert Prolog en même temps que le projet ce qui a mené à des problèmes de qualité de code.