Le Khan
Projet Prolog - IA02

Vincent Baheux Guillaume Hillion

Printemps 2016

Table des matières

1 Introduction

Dans le cadre de l'UV IA02, dont le programme est axé sur la programmation logique, nous devions réaliser un jeu de Khan fonctionnel en Prolog, respectant les règles du jeu et permettant des parties humain vs. humain, humain vs. machine et machine vs. machine.

Le Khan est un jeu de type abstrait. Il est composé d'un plateau de 6x6 cases ; chaque case possède une valeur de 1 à 3.

Lors du début de partie, le premier joueur choisit le côté du plateau (2 lignes les plus proches d'un côté) sur lequel il souhaite commencer, puis place sa Kalista (reine) et ses pions (sbires) sur son côté ; son adversaire fait de même sur le côté opposé.

Le but du jeu est de capturer la Kalista adverse en respectant les règles suivantes :

- Chaque pièce jouée doit se déplacer d'un nombre de cases égal à la valeur de la case de départ.
- Le joueur ne peut déplacer une pièce sur une case où il en possède déjà une autre ; de plus, un mouvement ne peut s'effectuer en passant pardessus une pièce.
- La capture d'une pièce adverse s'effectue en déplaçant une pièce sur la case où se situe la pièce adverse.
- Une fois une pièce jouée, celle-ci est coiffée d'une pièce appelée Khan ; l'adversaire, lors de son coup suivant, sera alors obligé de déplacer une pièce se situant sur une case de même valeur que le Khan.
- Lors du premier coup, le premier joueur peut déplacer la pièce qu'il souhaite, vu que le Khan n'est pas encore en jeu.
- Si aucun pion ne respecte la contrainte du Khan, ou si les pions la respectant ne peuvent pas se déplacer, le joueur peut alors soit déplacer le sbire qu'il souhaite, soit remettre un sbire capturé en jeu sur une case de même valeur que le Khan, auquel cas il ne pourra pas se déplacer durant ce tour.

Nous allons, dans un premier temps, exposer les structures de données et les prédicats principaux que nous avons utilisé, avant de décrire l'interface utilisateur du jeu ; nous finirons par évoquer les difficultés que nous avons rencontré, et proposer des améliorations.

2 Structures de données

2.1 Le plateau

Le plateau est une liste de lignes, chaque ligne étant elle-même une liste de tuples (Val, Type), où Val est la valeur de la case (de 1 à 3), et Type est le type de pion qui s'y trouve : kr, r1..r5 pour les rouges ; ko, o1...o5 pour les ocres et b si la case est vide.

En début de jeu, on a :

```
Plateau = [[(2, b),(3, b),(1, b),(2, b),(2, b),(3, b)],
           [(2, b),(1, b),(3, b),(1, b),(3, b),(1, b)],
           [(1, b),(3, b),(2, b),(3, b),(1, b),(2, b)],
           [(3, b),(1, b),(2, b),(1, b),(3, b),(2, b)],
           [(2, b),(3, b),(1, b),(3, b),(1, b),(3, b)],
           [(2, b),(1, b),(3, b),(2, b),(2, b),(1, b)]]
```

2.2 Les pièces

Une pièce est définie par un tuple pion(IdPion, Col, Lin, Etat, Val), où :

- IdPion est l'identifiant unique de la pièce (kr, r1..r5 pour les rouges ; ko, o1...o5 pour les ocres).
- (Col, Lin) est la position de la pièce sur le plateau.
- Etat détermine si la pièce est en jeu, si elle est en jeu et porte le Khan, ou si elle est hors jeu ; Etat peut prendre les valeurs 'in', 'khan' ou 'out'.
- Val est la valeur de la case où se situe le pion. On la récupère lors du placement du pion sur le plateau ; on peut ainsi gérer plus facilement la contrainte du Khan.

Lors de la recherche du meilleur mouvement, les pièces peuvent être passées sur le prédicat miniMaxPion(IdPion, Col, Lin, Etat, Val), un prédicat "fantôme" qui permet de transmettre la même information mais sans utiliser le prédicat pion, dont l'utilisation lors des simulations pourrait causer des erreurs dans la gestion des pions.

2.3 Le Khan

Comme nous venons de le voir dans la partie 2.2, le Khan est un état possible de l'élément Etat de la structure Pion ; une pièce porte le Khan lorsque son état est 'khan'. La contrainte d'unicité du Khan est gérée par la boucle principale du jeu ; à chaque tour, on retire le Khan du pion qui le portait, puis on met à jour la pièce qui vient d'être jouée afin qu'elle soit porteuse du Khan.

2.4 Mouvement

Un mouvement est défini par un tuple (Col1, Lin1, Col2, Lin2), où (Col1, Lin1) est la position de départ de la pièce à déplacer, et (Col2, Lin2) la position d'arrivée.

3 Prédicats principaux

3.1 play

Il s'agit du prédicat d'exécution du jeu. Dans un premier temps, il appelle initPlayers, le prédicat d'initialisation des types de joueurs (humain/machine). Il appelle ensuite initBoard(?Board), qui effectue l'initialisation des pions et renvoie le résultat dans un plateau. On entre ensuite dans la boucle de jeu principale, avec le prédicat main(+Board) ; cette boucle s'arrête dès qu'une Kalista est prise. Elle asserte alors un prédicat winner(+Player, +Type) afin de récupérer le gagnant dans play. On affiche le gagnant, et l'exécution est terminée.

3.2 initBoard(?Board)

Il s'agit du prédicat d'initialisation du plateau. Dans un premier temps, il remet à zéro toutes les assertions de l'exécution précédente (resetPions). On récupère ensuite le plateau vierge, déclaré dans un prédicat etatInitial (+Plateau). On initialise ensuite les couleurs rouge, puis ocre, en assertant l'information dans un prédicat dynamique getCote(Cote, Couleur).

Lors de l'initialisation de la couleur rouge, on appelle d'abord un prédicat de détermination du côté : getCote si le joueur est humain, randomCote si c'est une machine ; randomCote choisit un côté au hasard, tandis que getCote récupère un côté entré par l'utilisateur.

Ensuite, on appelle pour chaque joueur placerPions(+InBoard, +Cote, +Colour, −OutBoard), qui demande à l'utilisateur les positions de chaque pion si c'est un humain, et qui place les pièces aléatoirement sinon. Lorsque l'on place une pièce, on indique son identifiant sur le plateau, mais on asserte également un prédicat pion comme déclaré dans la partie 2.2.

3.3 main(+Board)

Il s'agit de la boucle principale du jeu. On récupère d'abord le statut de chaque joueur (humain ou machine) par unification avec un prédicat player (?Statut, +Couleur) pour chaque joueur. On rentre ensuite dans la boucle principale. On appelle alors le prédicat de tour playTurn si le joueur est humain, playTurnAI sinon. On récupère ensuite le côté du joueur courant (déclaré dans le prédicat getCote(Cote, Colour), afin d'afficher le plateau. On appelle ensuite de nouveau la boucle principale pour le joueur adverse, avec le plateau mis à jour.

La boucle principale s'arrête lorsqu'une Kalista a été prise, c'est-à-dire que son attribut Etat a été mis à jour à out. On récupère ainsi le gagnant, qui est le joueur de l'équipe opposée à la Kalista hors jeu.

3.3.1 Tour humain : playTurn(+InBoard, +Player, ?OutBoard)

Ici, on affiche tout d'abord le plateau au joueur. On appelle ensuite le prédicat influenceKhan(Colour), qui affiche au joueur quelle est la valeur

de la case où se situe le Khan, et s'il peut le respecter ou non. On appelle ensuite initMove(+Colour, ?Move, ?Pion), qui initialise le mouvement du joueur, en récupérant l'identifiant du pion à déplacer, ainsi que la position d'arrivée de celui-ci - on traite au passage les cas d'erreur. On appelle ensuite possibleMoves(+Board, +Colour, −MoveList) afin de comparer, par la suite, le mouvement saisi par l'utilisateur avec la liste de mouvements possibles pour s'assurer de la validité du mouvement saisi.

On appelle alors execMove(+InBoard, +Colour, +Move, +Pion, +MoveList, ?OutBoard), qui vérifie que le mouvement est valide ; si c'est le cas, il effectue le transfert du pion à l'aide du prédicat transfert(+InBoard, +Move, +Pion, ?OutBoard). Sinon, il relance le tour de jeu.

3.3.2 Tour machine : playTurnAI(+InBoard, +Player, ?OutBoard)

Ici, on appelle generateMove(+InBoard, +Colour, −Move), qui génère le meilleur mouvement possible pour l'ordinateur, considérant un état du plateau donné et le joueur courant. Une fois ce mouvement obtenu, on l'effectue à l'aide du prédicat transfert(+InBoard, +Move, +Pion, ?OutBoard), puis on renvoie le nouveau plateau.

3.4 transfert(+InBoard, +Move, +Pion, ?OutBoard)

Ce prédicat transfert une pièce d'une case à un autre ; s'il y a une pièce adverse sur la case, il modifie son statut en 'out', et la remplace par la nouvelle pièce. Le prédicat enlève également le Khan de la pièce précédente, pour la réaffecter à la pièce qui vient d'être déplacée. Si la pièce était hors plateau, elle est replacée sans problème, vu qu'on considère qu'une pièce sorti du plateau est en position (0, 0).

3.5 possibleMoves(+Board, +Player, -PossibleMoveList)

Ce prédicat explore l'ensemble des mouvements possibles, et renvoie ces derniers sous forme de liste. Si un pion est hors-jeu et que le Khan ne peut pas être respecté, on rajoute sa remise sur le plateau dans la liste des mouvements possibles ; on rajoute également tous les mouvements possibles dans ce cas là.

Sinon, on explore tous les mouvements possibles pour les pions respectant la contrainte du Khan ; on avance case par case, en ne revenant pas en arrière (on ne peut pas passer 2 fois par la même case). Si on tombe sur une pièce en cours de route, ou sur une pièce de même couleur que le joueur actuel en cours de mouvement, le mouvement n'est pas valide ; on ne le retient donc pas.

Si aucune règle n'interfère avec l'exploration jusqu'à la position terminale du mouvement, celui-ci est considéré comme valide ; on l'ajoute donc à la liste des mouvements possibles.

3.6 generateMove(+Board, +Player, -Move)

Le prédicat generateMove(+Board , +Player , −Move) permet au programme de rechercher dans la liste des tuples (heuristique, Move) la mouvement avec l'heuristique la plus élevée.

Nous aurions souhaité réaliser cette recherche à l'aide de l'algorithme de recherche miniMax α β. Bien que nous ayons bien compris les règles de fonctionnement de cet algorithme, en particulier grâce à des vidéos Youtube de l'Université de Berkeley (https ://www.youtube.com/watch ?v=xBXHtz4Gbdo), il a été difficile de mettre en place cet algorithme pour ce programme. Notre travail sur l'algorithme α β n'est donc pas intégré au projet mais consultable et commenté dans le fichier alphabeta2.pl.

3.7 heuristic(+Board, +Player, -Value)

En mode "machine", l'heuristique se base sur les 7 prédicats suivants ;

— nbSbiresAlliesenJeu qui calcule le nombre d'Allies en Jeu, le but est de les avoir tous en jeu

— nbSbiresEnnemisEnJeu qui calcule le nombre d'ennemis en jeu, et cherche dans l'idéal à stabiliser ce nombre autour de 4 pions

— distanceSbiresKalista calcule le nombre de sbires alliés dans un rayon de trois cases autour de la Kalista ennemie

— defenseKalistaAlliee et defenseKalistaEnnemie calculent le nombre de pions bloquant l'accès aux kalistas, le but du jeu étant d'en avoir maximum 2 pour soi et 0 pour l'adversaire

— les prédicats gagne et perdu qui renvoient respectivement 100 et 0 en cas de victoire de l'un des joueurs

4 Interface utilisateur

L'affichage et les entrées du jeu se font intégralement en console.

4.1 Initialisation

Tout d'abord, on demande à l'utilisateur quel type il souhaite affecter à chaque joueur (`humain` ou `machine`).

On affiche ensuite le plateau vide, afin de permettre à l'utilisateur de visualiser ce dernier.

Le joueur rouge choisit le côté du plateau sur lequel il souhaite commencer ; si c'est une machine, on choisit le côté aléatoirement. Si c'est un humain, on lui demande de saisir le côté qu'il souhaite (`gauche`, `droite`, `haut`, `bas`).

Ensuite, chaque joueur doit placer ses pièces sur le plateau ; si c'est une machine, on place les pièces automatiquement de manière aléatoire.

Si le joueur est un humain, on demande à l'utilisateur la coordonnée sur laquelle il souhaite placer un pion pour la kalista et les 5 sbires. Les coordonnées vont de `a1` à `f6`. Si la case saisie est déjà occupée par un autre pion ou si elle ne se situe pas du bon côté, un message d'erreur s'affiche et l'utilisateur doit rentrer à nouveau la coordonnée de cette pièce.

A chaque placement de pion pour le joueur, et à la fin de l'initialisation pour la machine, on affiche l'état du plateau.

4.2 Boucle principale

Lors de l'exécution de la boucle principale, on indique à chaque tour quel est le joueur qui doit jouer. Si le joueur est humain, on affiche tout d'abord le plateau de jeu de son côté ; on lui indique ensuite quelle est la valeur du Khan, puis on lui demande quel pion il souhaite déplacer, ainsi que la position d'arrivée de ce pion. S'il rentre une valeur erronnée ou si le mouvement n'est pas valide, on recommence la saisie.

Si le joueur est une machine, on affiche le plateau une fois le pion déplacé, du point de vue de la machine.

4.3 Fin de jeu

Dès que l'une des deux Kalistas est prise, on sort de la boucle principale et on renvoie le joueur qui a pris la Kalista ; on affiche ainsi à l'utilisateur le gagnant, s'il était humain ou machine, puis l'exécution se termine.

5 Problèmes rencontrés

5.1 Affichage du plateau

Lors du développement de l'initialisation du plateau, nous nous sommes posés la question suivante : comment faire pour gérer les rotations de plateau selon l'endroit où se situe le joueur, afin qu'il puisse avoir une vue similaire à celle qu'il aurait lors d'une partie réelle ?

Pour cela, nous avons envisagé deux solutions différentes : Soit nous pouvions gérer cela avec 4 plateaux différents, chacun ayant les valeurs de cases correspondant à une rotation particulière ; soit nous pouvions n'utiliser qu'un seul plateau, et gérer l'affichage différemment selon le côté choisi.

Compte tenu du fait que le nombre d'opération afin d'effectuer le placement (ou le déplacement) d'un pion sur le plateau aurait été augmentée, nous avons opté pour la seconde solution. Le traitement des différents cas d'affichage était assez complexe à développer, car il fallait gérer les quatre cas possibles, et que chacun d'entre eux différait totalement des autres dans sa structure. Nous avons finalement réussi à implémenter cet affichage avec succès.

5.2 Algorithme de recherche du meilleur mouvement

Nous avons dû nous limiter à une simple recherche du meilleur mouvement parmi la PossibleMoveList étant donné la difficulté de l'algorithme minimax Alpha Beta.

6 Améliorations possibles

6.1 Erreurs de saisie

Certaines erreurs de saisie ne sont pas gérées, e.g. l'initialisation du joueur (humain ou machine) ; de plus, certaines saisies provoquent de la casse, et font échouer l'exécution du jeu. Nous pensons qu'une interface graphique avec interactions par clics serait la meilleur solution pour éviter ce genre de problèmes.

6.2 Interface graphique

L'affichage du jeu se fait actuellement par console ; l'idéal serait d'implémenter une interface graphique, afin que les interactions entre les joueurs humains et le jeu soient plus « *user-friendly.* »

6.3 Placement des pions de l'IA

Actuellement, l'IA du jeu place aléatoirement les pions sur le plateau. Nous pouvons améliorer cela grâce aux règles suivantes :

— La Kalista doit être placée sur la ligne au bord du plateau ;
— Les pions doivent être placés de façon à alterner régulièrement entre les deux lignes de placement possibles ; ainsi, les pions ne seront placés que sur 2 valeurs de cases différentes, et il sera alors plus facile de désobéir au Khan.

6.4 Optimisation de l'algorithme de recherche du meilleur mouvement

Nous avons tenté d'implémenter un algorithme Alpha Beta, sans succès ; nous nous sommes donc arrêtés à une recherche simple du meilleur mouvement, sans explorer les mouvements suivants dans l'arbre des possibilités ; une amélioration possible serait de terminer l'implémentation de cet algorithme.