Félix ALIÉ
Hélène ARMAOS

IA02 – P16

# Programmation en Prolog d'un Jeu de KHAN

# Sommaire

# Introduction

Le but de ce projet est d'implémenter le jeu de KHAN en utilisant PROLOG, afin de permettre une partie sous plusieurs modes différents :

- à deux joueurs humains ;
- à un joueur humain contre un joueur simulé virtuellement par le programme.

Le jeu de KHAN se joue à deux sur un tapis de jeu comportant 6 lignes de 6 cases, qui ont chacune un poids de 1, 2 ou 3. Chaque joueur a une Kalista et cinq sbires de couleur rouge ou ocre selon le joueur. Le but du jeu est d'éliminer la Kalista adverse, en la rejoignant sur sa case. Le joueur « Rouge » choisit un bord parmi les 4 bords possibles et installe ses pièces sur les deux premières rangées, le joueur « Ocre » fait de même. Les pièces peuvent se déplacer en avant, en arrière latéralement, et tourner à angle droit. Elles ne peuvent pas passer sur une case occupée et passer deux fois par une même case, mais peuvent finir leur coup sur une case occupée par une pièce adverse. Dans ce cas-là, la pièce adverse sort du jeu.

Il y a aussi présence du KHAN qui a pour rôle d'influencer le déplacement des pièces, en coiffant la pièce qui vient de bouger. Le joueur suivant, doit alors jouer une pièce occupant le même type de case que celle sur laquelle est la pièce coiffée. Les joueurs ne peuvent pas toujours obéir au KHAN, dans ce cas-là, ils peuvent bouger une autre pièce ou remettre en jeu un des sbires précédemment sortis du jeu par l'adversaire.

# Prédicats principaux du jeu

## Organisation du programme

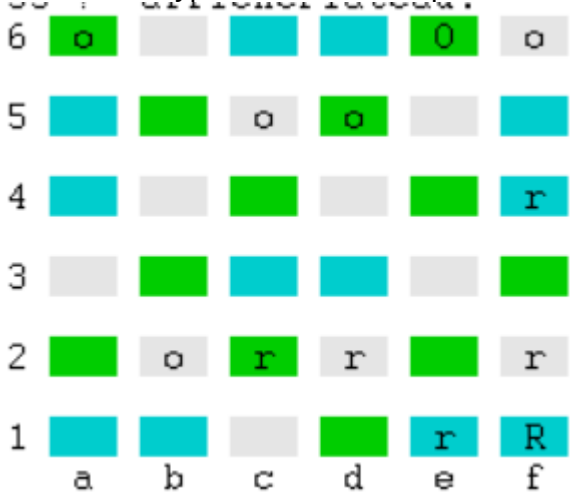
Le programme est réparti en 6 sous fichiers différents afin de se retrouver au mieux dans un code Prolog parfois difficile à relire. Ces six fichiers sont donc ainsi respectivement dédiés à l'initialisation, à l'affichage, au contrôle du jeu, au contrôle des tours, aux services, et au calcul des mouvements.

## Représentation du plateau

Afin de représenter le plateau de jeu nous avons fait le choix d'une liste de six sous-listes composées chacune de six éléments. Nous avons implémenté le plateau originel avec les poids 1, 2 et 3 initiaux. En fonction du côté choisi par le joueur « Rouge », nous avons utilisé quatre prédicats différents qui modifie la valeur de notre dynamic plateau/1, afin de visualiser quatre plateaux différents en fonction du côté choisi. Par exemple, si le joueur « Rouge » choisit le côté droit, cela va donner :

```
[[2,2,1,3,2,2],
 [3,1,3,1,3,1],
 [1,3,2,2,1,3],
 [2,1,3,1,3,2],
 [2,2,1,3,1,2],
 [3,1,2,2,3,1]]
```

Nous avons pour l'affichage décidé d'utiliser les librairies d'affichage proposées par SWI Prolog permettant de visualiser les types des cases simplement à l'aide de leur couleur.




Les cases de type 1 sont blanches, les 2 bleues, et les type 3 sont vertes. On utilise également un système de coordonnées en lettre + chiffre que nous détaillerons plus loin, dans le but de faciliter l'usage de notre programme. L'ensemble de l'affichage est géré dans un fichier à part (affichage.pl) au travers du prédicat principal affichePlateau.

Afin de faciliter et décomposer le problème nous avons décidé d'isoler les prédicats dites de traitement/manipulation (de service) des listes et les prédicats composant le cœur du programme.

## Services

Afin de faciliter le développement de prédicats généraux, nous avons développé un grand nombre de plus petits prédicats regroupés dans le fichier service.pl. Ces derniers peuvent être eux-mêmes regroupés en 4 grands groupes de prédicats :

- Prédicats de manipulation des pièces : ensemble des prédicats pour déplacer les pièces, pour manipuler le khan, pour reset les pions etc…

- Prédicats de service pour chaque tour : ensemble des prédicats permettant de prendre des décisions lors du déroulement d'un tour.

- Prédicats de manipulation des listes : ensemble de prédicats génériques permettant de manipuler des listes, inversion, duplication, supression d'éléments etc.

- Predicats de parsing : transforme les coordonnées du type paire d'entier (chiffre, chiffre), en coordonnées alphanumériques, ou inversement, en fonction des besoins du programme.

# Prédicats principales du programme

Le déroulement principal du jeu se fait de la façon suivante :

Une partie de jeu = initialisation des types de joueurs (humain ou I.A) -> choix plateau et placement des pièces -> test du Khan -> pions jouables -> coups possibles -> choix du coup par le joueur1 -> test/validité du coup -> mouvement du pion -> test de la kalista -> pose du khan -> joueur2 joue ->... -> kalista prise -> fin de la partie .

Ces prédicats sont le cœur de notre programme. Elles font appel aux prédicats de service pour les tâches basique de manipulation des valeurs.

## Prédicats d'initialisation

C'est le prédicat init, qui permet l'initialisation du type de joueur (si humain ou non) et du plateau de jeu selon ce mode de fonctionnement :

- choisir type joueur
- choisir type joueur 2

Si J1 humain :
- Demander quel côté du plateau (haut, bas, droit ou gauche)
- Demander le placement de la Kalista rouge
- Demander le placement des Pions rouges

Sinon
- le côté du plateau est choisi aléatoirement
- le placement de la Kalista rouge et des Pions rouges sont aléatoires à partir d'une liste de faits de positions possibles pour les rouges.

Si J2 humain :
- Demander le placement de la Kalista ocre
- Demander le placement des Pions ocres

Sinon
- le placement de la Kalista ocre et des Pions ocres sont aléatoires à partir d'une liste de faits de positions possibles pour les ocres.

Ensuite, les coordonnées entrées par le joueur humain sont testées, pour savoir si elles sont valides, afin de procéder à l'initialisation.
Les pieces du joueur non humain sont placées automatiquement par les prédicats placementOrdi. A chaque placement d'un pion ou d'une kalista, on va retirer la case choisie aléatoirement de la liste des positions possibles. Le plateau choisi est initialisé, en fonction du côté choisi par le joueur humain ou initialisé aléatoirement.

Les coordonnées sont lues sous la forme [a-z]\ d et parsée afin de récupérer les positions X et Y. Par exemple, le joueur rentre a1, cela va être parsé et transformé en (1,1).

## Tour humain

La fonction tourhumain permet de faire les fonctionnalités suivantes :

- Tester si le khan est posé
- Évaluer les pions jouables
- Calculer et proposer les mouvements possibles
- Le joueur choisit où bouger son pion
- Tester si la Kalista adverse est prise
- Pose du Khan
- Tour suivant, l'autre joueur va jouer

Un sous prédicat de tourhumain va permettre de déterminer si le joueur peut obéir au khan ou non et gérer en fonction les cas possibles :

- si le khan n'est pas encore en jeu
- si le khan ne permet pas au joueur d'obéir
- si le joueur a des pions sur des cases du même type que le khan
- si le joueur a des pions sur des cases du même type que le khan mais qu'il ne peut pas les jouer.

On distingue deux prédicats : proposeCoupsSansKhan et proposeCoups. Si le joueur ne peut obéir au khan, proposeCoupsSansKhan va être appelé et proposer soit une pièce déjà éliminée soit de bouger une pièce qui n'est pas sur le type de case du khan.

La fonction checkMouvement vérifie s'il y a mouvement d'une pièce, si les coordonnées entrées par le joueur sont valides. Elle va aussi tester si un pion adverse est mangé et si ce pion est la Kalista, ce qui terminerait le jeu.

## Moves – mouvement

Le prédicat calculeMouvements va permettre de calculer tous les mouvements possibles d'un pion, en fonction du type de case sur lequel ce dernier se trouve.

On va d'abord utiliser le prédicat getSucc qui va déterminer tous les successeurs possibles d'une case (c'est-à-dire tous les voisins possibles d'une case où un pion peut se déplacer d'un coup).

Puis, le prédicat getAllSucc va déterminer en fonction du nombre de coups possibles pour une pièce (1, 2 ou 3), tous les successeurs possibles hormis les cases où il y a déjà des pions adverses, sauf si c'est le dernier coup, et dans ce cas-là, le pion adverse peut être mangé.

## Main – Start

La fonction Start permet au joueur humain de jouer, elle va appeler l'initialisation des structures différentes et va permettre le déroulement de la partie jusqu'à la victoire.

# Choix des structures de données

Pour la représentation du plateau ainsi que des pièces, nous avons utilisé des structures dynamic. En effet, le plateau peut se présenter sous 4 formes, en fonction du côté choisi par le joueur « Rouge », au début du jeu. Cela semble naturel pour les pièces, puisqu'on met en arguments leurs coordonnées X et Y, dans le plateau, ce qui est amené à changer au cours de la partie, lors de leurs déplacements.

Pour la visualisation des poids 1, 2 et 3 ainsi que des pièces, sbire rouge ou ocre et kalista rouge ou ocre, nous avons choisi d'utiliser la fonction ansi_format/3, qui permet d'implémenter un symbole de couleur sur un fond de couleur. Cela permet de suivre la partie à tout moment, avec un code couleur simple et représentatif.

# Intelligence Artificielle

Nous n'avons pas implémenté d'IA par manque de temps, mais nous avons donné l'heuristique.
Si l'on peut tuer Kalista adverse, alors on tue.
Sinon on cherche dans les successeurs des successeurs si on peut se rapprocher à n-1 coup de manger la Kalista adverse.
Si on en trouve un, on teste si le mouvement nous met en danger :
Si le Khan est de type T
    Si Kalista est de couleur T
        Si l'adversaire ne desobeit pas au khan, executer mouvement
        Sinon tester coups ennemis pour detecter danger
            Si non danger, executer mouvement
            Sinon essayer autre coup
    Si Kalista est de type autre que T
        Calculer mouvements adverses si le nouveau type du khan est T
            Si non danger, executer mouvement,
            Sinon essayer coup entrainant autre Type.
    Sinon Move aléatoire parmi les move restants.

# Difficultés et améliorations

**Points forts :**

- L’aspect visuel et l’interface, relativement agréable à utiliser

**Améliorations et difficultés :**

- Nous n’avons pas implémenté d’intelligence artificielle mais seulement l’heuristique par manque de temps
- Il nous a fallu du temps pour nous approprier la logique du Prolog
- Gérer les conditions if... else ainsi que les boucles est peu intuitif sous Prolog
- Utilisation parfois de red cut
- Toutes les exceptions ne sont pas gérées
- Et il reste des bugs