RAPPORT DE PROJET – IA02

Jeu de KHAN

Lila Sintes

Marion Chan-Renous-Legoubin

Table des matières

I. DESCRIPTION DU JEU

Le jeu de KHAN est un jeu composé d'un plateau de 6*6 cases, possédant chacune une valeur numérique allant de 1 à 3, et de 12 pions : 5 sbires et 1 Kalista, de couleur ocre ou rouge, et un accessoire, le KHAN. Le but du jeu est d'éliminer la Kalista inverse, en la rejoignant sur sa case avec l'un de ses pions.

Déroulement d'une partie :

Le joueur rouge oriente le plateau de jeu parmi les quatre bords possibles et positionne ses pions sur les 12 cases situées dans les deux rangées les plus proches de lui. Le joueur ocre positionne ensuite ses pions sur les deux rangées les plus proches de lui.

A chaque tour de jeu, le joueur rouge commençant, le joueur déplace un pion, en fonction de la valeur de la case sur laquelle se trouve celui-ci. Les règles de déplacement sont telles qu'il est interdit de passer au-dessus d'une case occupée, de passer deux fois par la même case, ou de se déplacer en diagonale. A chaque tour de jeu, le KHAN "coiffe" la pièce venant d'être jouée. Le joueur adverse doit ensuite déplacer un pion qui se trouve sur une case possédant la même valeur que la case où se trouve le pion coiffé par le KHAN. Si aucune pièce ne satisfait cette règle, ou qu'aucun mouvement n'est possible depuis une case possédant cette valeur, il est possible de ramener un pion dans le jeu qui a été éliminé précédemment par le joueur adverse, ou déplacer une autre pièce de son choix.

Dans le cadre de ce projet, il nous a été demandé d'implémenter, en utilisant PROLOG, ce jeu, avec trois modes de jeu : Humain vs Humain, Humain vs Ordinateur et Ordinateur vs Ordinateur.

II. STRUCTURE DES DONNEES

1. Plateau de jeu

Nous avons choisi de modéliser le plateau de jeu grâce à une liste de six éléments, dont chacun de ces éléments est également une liste, composée de 6 chiffres correspondant à la valeur de la case, afin d'obtenir un tableau 6x6.

Afin de modéliser le fait que le joueur peut choisir une orientation pour le plateau de jeu, nous avons créé quatre prédicats "**tab**" possédant une liste différente en argument, correspondant aux quatre orientations possibles, donnant lieux à quatre plateaux de jeu différents.

Nous avons choisi de créer un prédicat dynamique "**board**", qui possède comme argument une liste, afin de pouvoir gérer dynamiquement le plateau du jeu en fonction du choix du joueur.

Concernant la partie graphique, nous avons choisi de créer 4 autres prédicats "**tabg**", comportant les mêmes éléments que les prédicats tab, mais mis en forme d'une telle façon qu'il est possible de modifier la valeur d'un des éléments de la liste tout en gardant une forme de tableau adéquate. Nous avons donc également créé un autre prédicat dynamique "**showBoard**", destiné à gérer la partie graphique du jeu, qui sera indépendante du prédicat "**board**".

2. Modélisation des joueurs et de leurs différents pions

Afin de modéliser les deux joueurs, nous avons décidé de définir un prédicat dynamique "**joueur**", permettant de savoir quel joueur est en train de jouer en fonction de son argument, qui prend les valeurs "rouge" ou "ocre".

Afin de modéliser chaque pion, nous avons choisi de créer un prédicat dynamique "**pion**", possédant 6 arguments. Le premier argument correspond à la couleur du pion (ocre ou rouge), le deuxième au numéro du pion (0 pour la khalista, et un chiffre entre 1 et 5 pour les sbires), le troisième correspond à la ligne du tableau où se trouve le pion, le quatrième correspond à la colonne du tableau où se trouvent le pion, le cinquième correspond à si le khan coiffe cette pièce ou non: si le khan le coiffe cet argument a pour valeur 'Khan', sinon il s'agit d'une variable muette '_', et le dernier argument est le type du pion (sbire ou khalista).

Le prédicat dynamique "**listPions**" permet de stocker une liste comportant toutes les coordonnées des différents pions du jeu à chaque instant du jeu.

Afin de modéliser le Khan et de pouvoir récupérer la valeur de la case de la pièce qu'il coiffe, nous avons choisi de créer un prédicat dynamique "**khan**", qui possède un argument correspondant à la valeur numérique de la case (entre 1 et 3).

III. PRINCIPAUX PREDICATS

1. Prédicats généraux

Tout d'abord, nous avons un prédicat "**boucle_menu**", qui permet de démarrer le programme en affichant le menu du jeu et qui invite le joueur à faire un choix entre les différents modes de jeu : humain vs humain, humain vs ordinateur, ordinateur vs ordinateur, ou quitter le programme. En fonction du choix saisi par le joueur, un prédicat "**init**" est appelé. Celui-ci est différent pour chaque mode de jeu choisi.

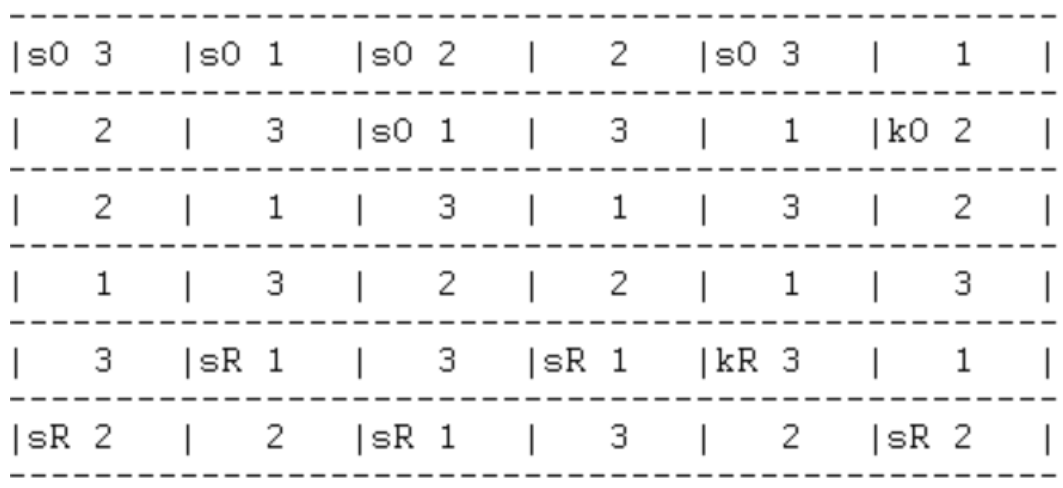
Afin de choisir l'orientation du tableau, nous avons créé un prédicat "**chooseBoard**". Ce prédicat permet de modifier dynamiquement le prédicat "**board**" afin de lui affecter la liste correspondant à l'orientation choisie. Ensuite, le plateau de jeu est affiché à l'écran.

La partie graphique est gérée par plusieurs prédicats. Le prédicat "**printLign**", permet l'affichage d'une ligne du tableau, chaque ligne étant elle-même une liste, et le prédicat "**printTab**", permet l'affichage du tableau entier, en appelant successivement le prédicat "**printLign**". Le prédicat "**printGraphism**" permet ainsi de récupérer la liste du prédicat dynamique "**showBoard**" et de l'afficher en console à l'aide du prédicat "**printTab**".

```
-------------------------------------------------
|   1   |   2   |   2   |   3   |   1   |   2   |
-------------------------------------------------
|   3   |   1   |   3   |   1   |   3   |   2   |
-------------------------------------------------
|   2   |   3   |   1   |   2   |   1   |   3   |
-------------------------------------------------
|   2   |   1   |   3   |   2   |   3   |   1   |
-------------------------------------------------
|   1   |   3   |   1   |   3   |   1   |   2   |
-------------------------------------------------
|   3   |   2   |   2   |   1   |   3   |   2   |
-------------------------------------------------
```

*Plateau de jeu initial affiché grâce au prédicat "**PrintGraphism**"*

Afin de modifier l'affichage du plateau de jeu après chaque tour de la partie, nous avons créé un prédicat "**replaceGraphism**". Celui-ci se décline en plusieurs versions en fonction du type de pion et du type de case. Le prédicat "**modifyGraphism**" permet quant à lui d'appeler le prédicat "**replaceGraphism**" et de modifier la liste présente dans le prédicat dynamique "**showBoard**".



Exemple de plateau de jeu affiché suite à l'initialisation de tous les pions

Un prédicat "**occupied**" permet de déterminer si les coordonnées du tableau passées en argument sont occupées par un pion ou non.

Un prédicat "**choosePlace**" demande à l'utilisateur de rentrer une valeur de ligne et de colonne, afin de pouvoir initialiser les coordonnées d'un pion, de les modifier durant le jeu etc.

Le prédicat "**choosePion**" demande à l'utilisateur de rentrer le numéro du pion (entre 0 et 5) afin de le déplacer.

Le prédicat "**possibleMoves**" génère une liste de tous les coups possibles d'un joueur par rapport aux règles du jeu. Il prend en compte toutes les règles du jeu (règle du KHAN, règles sur les mouvements).

Nous avons défini un prédicat "**changer_joueur**" qui permet de changer la valeur du prédicat dynamique "**Joueur**" en la valeur du joueur inverse. Ainsi, si le joueur actuel est le joueur rouge, l'appel à ce prédicat permettra de changer la valeur du prédicat en 'ocre'.

Le prédicat "**isFinale**" permet de vérifier si le tour joué était le dernier : si une Kalista a été éliminée ou non.

Le prédicat "**eatPion**" permet d'éliminer du plateau de jeu un pion adverse, en modifiant dans le prédicat dynamique '**pion**' les valeurs des coordonnées du pion mangé. Si le pion est mangé, ses coordonnées prennent la valeur (0,0).

Le prédicat "**setPion**" permet de changer les valeurs des coordonnées des pions dans le cadre d'un déplacement dans le prédicat dynamique "**pion**". Ainsi, ce prédicat traite les trois cas possibles lors d'un déplacement :

- Si la case d'arrivée est occupée par un pion adverse (prédicat "**occupied**"), le prédicat '**eatpion**' est appelé. Le pion est ensuite déplacé sur le plateau de jeu grâce au prédicat '**ReplaceGraphism**'. Ensuite, les coordonnées du pion déplacé sont modifiées dans le prédicat dynamique pion avec les nouvelles valeurs de sa position.
- Si la case d'arrivée n'est pas occupée, le pion est simplement déplacé.
- Si les coordonnées d'arrivée du pion ne sont pas valides, (testées avec le prédicat '**PossibleMoves**'), le joueur est invité à entrer des coordonnées valides.

Ce prédicat est décliné en deux autres versions "**setPionInit**" et "**setPionFirst**", qui permettent respectivement d'initialiser les coordonnées des pions et de modifier la valeur des coordonnées du premier pion déplacé lors d'une partie.

Le prédicat dynamique "**best**" permet de stocker les coordonnées du meilleur coup à jouer par l'ordinateur, afin de pouvoir appeler le prédicat "**moveBestCoup"** qui permet de déplacer la pièce concernée par le meilleur coup à jouer.

2. Fonctionnement du mode de jeu Humain vs Humain

Le prédicat correspondant à l'initialisation du jeu dans ce mode est appelé via le choix du menu (**init(hh)**). Le joueur choisi ensuite une orientation du tableau (prédicat "**chooseBoard**"), et le plateau initial est affiché à l'écran (prédicat "**printGraphism**"). Le prédicat "**boucle_init_rougeH**" est ensuite appelé. Ce prédicat sert à initialiser la position de chacun des pions rouges via un appel récursif au prédicat "**initialisation**". L'utilisateur est invité à entrer une valeur de ligne et de colonne pour chacun de ses six pions (prédicat "**choosePlace**"). Pour chaque pion, on procède ensuite à vérifier que les coordonnées rentrées sont valides : on vérifie que les valeurs sont dans les limites du plateau et dans les limites possibles pour le joueur (deux premières lignes du plateau), mais aussi que la case du plateau n'est pas déjà occupée par une autre pièce grâce à la négation du prédicat "**occupied**". Si les coordonnées entrées sont valides, le prédicat "**SetPionInit**" est appelé, qui initialise les coordonnées des pions dans le prédicat dynamique '**pion**'.

Après avoir initialisé les pions rouges, le prédicat "**boucle_init_ocreH**" est appelé. Celui-ci permet d'initialiser de la même manière les pions ocres.

Le prédicat **'playFirstMove'** est ensuite appelé. Nous avons choisi de faire un prédicat spécifique pour le premier tour de jeu, car durant ce tour le khan n'a pas encore de valeur, ainsi le joueur peut choisir n'importe quelle pièce à déplacer, contrairement aux autres tours de jeu. Ce prédicat est construit de la façon suivante :

- Le premier joueur (le rouge selon les règles du jeu) est invité à choisir un pion à déplacer (prédicat '**choosePion**')
- Le joueur entre les coordonnées de la case d'arrivée du pion (prédicat '**choosePlace**')
- Les coordonnées de la case d'arrivée sont vérifiées
- Si ces coordonnées sont valides, les coordonnées du pion choisi sont modifiées (prédicat '**SetPionFirst**'), sinon, le joueur est invité à entrer des coordonnées valides
- Une fois le pion déplacé, le joueur est changé en 'ocre' (prédicat **'changer_joueur'**).

Apres l'initialisation de tous les pions et le premier coup joué, le prédicat '**play(hh,0)**' est appelé. A chaque tour de jeu, ce prédicat affiche à l'écran : la couleur du joueur qui est en train de jouer, la valeur du khan (1, 2 ou 3), les coordonnées des différents pions du joueur, et demande au joueur d'entrer au clavier le numéro du pion qu'il souhaite déplacer (prédicats '**choosePion**' et '**choosePlace**"). Apres avoir entré un choix valide, il est demandé au joueur d'entrer les coordonnées d'arrivée du pion sur le plateau de jeu. Le prédicat **'SetPion'** est ensuite appelé.

Le prédicat '**isFinale**' est ensuite appelé. Si le tour joué est le dernier tour du jeu, le prédicat '**play**' est appelé avec comme valeur 1 pour le deuxième argument, ce qui provoque l'affichage du nom du gagnant et la fin de la partie. Si une kalista n'a pas été éliminée durant le tour de jeu en cours, le prédicat '**play**' est appelé avec comme valeur 0 pour le deuxième argument, ce qui lance un nouveau tour de jeu sur le même schéma que celui expliqué précédemment.

3. Fonctionnement du mode de jeu Humain vs Ordinateur

Le prédicat correspondant à l'initialisation du jeu dans ce mode est appelé via le choix du menu (**init(ho)**). Après le choix de l'orientation du tableau, l'initialisation des pions du joueur humain (rouge) est lancée via le même prédicat **'boucle_init_rougeH'** que dans le mode de jeu humain vs humain. Ensuite, l'initialisation des pions de l'ordinateur (ocre) est lancée via le prédicat **'boucle_init_ocreO'**. Ce prédicat fait appel au prédicat "**initialisationO**", qui est basé sur le même modèle que celui pour initialiser les pions d'un joueur humain (prédicat "**initialisation**"), mais au lieu de demander les coordonnées au joueur, les coordonnées sont générées aléatoirement grâce au prédicat prédéfini '**random**'. Le joueur humain joue ensuite le premier coup de la partie (prédicat "**playFirstMove**"). Le prédicat "**play(ho,0)**" est ensuite appelé. Ce prédicat est défini de deux manières différentes, si le joueur est le joueur rouge (humain), ou le joueur (ocre). Dans le cas du joueur humain, le tour de jeu est basé sur le même schéma que pour un tour de jeu dans le mode Humain vs Humain. Pour le tour de jeu de l'ordinateur, le prédicat **'iaMove**' est appelé. Celui-ci génère le mouvement que l'ordinateur va effectuer (explications de ce prédicat dans la partie IV). Les coordonnées du pion déplacé sont ensuite modifiées dans le prédicat dynamique '**pion**'.

Sur le même modèle que le mode de jeu Humain vs Humain, le prédicat **'isFinale**' est appelé, puis le prédicat '**play'** récursivement jusqu'à la fin de la partie.

4. Fonctionnement du mode de jeu Ordinateur vs Ordinateur

Le prédicat correspondant à l'initialisation du jeu dans ce mode est appelé via le choix du menu (**init(oo)**). L'orientation du plateau de jeu est choisie aléatoirement par ce prédicat (entre 1 et 4). L'appel aux prédicats permettant l'initialisation des positions des pions est ensuite effectué (prédicats "**boucle_init_rougeO**" et "**boucle_init_ocreO**"), qui sont les mêmes que ceux utilisés dans le mode de jeu humain vs ordinateur pour initialiser les pions de l'ordinateur. On stocke, grâce au prédicat prédéfini "**findall**', la liste des coordonnées des pions rouges dans une liste (List1) et ceux des pions ocres dans une autre liste (List2). Le prédicat **'playFirstMoveO**' permet d'appeler le prédicat **'iaMove'** pour le premier tour du jeu, lorsque le khan n'a pas encore été affecté à une valeur. Le joueur est ensuite changé (prédicat "**changer_joueur**"). A chaque tour de jeu, la valeur du khan est affichée, ainsi que la couleur du joueur qui est en train de jouer. Les tours de jeu se succèdent automatiquement sans que l'utilisateur n'ai besoin d'entrer quoique ce soit au clavier, jusqu'à ce qu'un des deux joueurs gagnent (prédicat '**isFinale**').

IV. DESCRIPTION DE L'INTELLIGENCE ARTIFICIELLE

Afin de modéliser l'intelligence artificielle présente dans les modes de jeu comportant au moins un joueur de type "Ordinateur", nous avons choisi d'implémenter différents prédicats. Le but final étant que l'ordinateur fasse le meilleur mouvement de pion possible étant donné l'état actuel du jeu.

Tout d'abord, nous avons choisi d'initialiser les pions du joueur ordinateur de façon aléatoire sur le plateau de jeu. En effet, nous avions pensé à établir une position initiale "stratégique" des pions (par exemple en positionnant la Kalista dans un coin du plateau et en l'entourant de sbires afin qu'elle soit plus difficilement atteignable). Cependant, avec une position initiale prédéfinie, les parties du mode de jeu "Ordinateur vs Ordinateur" auraient été les mêmes à chaque partie, et cela nous a paru comme ayant un intérêt moindre.

Nous avons défini un prédicat **'ListerPions'**, qui nous permet d'obtenir en temps réel les coordonnées de tous les pions du joueur à chaque tour de la partie. Nous utilisons le prédicat dynamique "**listPions**" afin de stocker cette liste.

Nous avons ensuite défini un prédicat **'PossiblePionsList'** qui liste les pions pouvant être déplacés à chaque instant t du jeu. Ce prédicat utilise la liste résultante de l'appel au prédicat '**PossiblesMoves**', qui renvoie une liste de tous les mouvements possibles pour le joueur.

Le prédicat principal de l'IA de ce jeu réside dans le prédicat '**BestCoup**'. Ce prédicat réutilise la liste résultante du prédicat **'PossiblePionsList'**. A partir de cette liste, ce prédicat permet de renvoyer les coordonnées du meilleur coup à jouer. En effet, celui-ci commence par trouver les coordonnées de la kalista du joueur adverse, grâce au prédicat **'autre_joueur'**, et au prédicat dynamique '**pions**'. Ce prédicat liste ensuite les coordonnées des pions adverses à l'aide du prédicat '**listerPions**'. Afin de chercher le meilleur coup à effectuer, nous avons adopté le raisonnement suivant :

Nous calculons la distance entre les pions du joueur adverse et la kalista du joueur en train de jouer, et la distance entre les pions du joueur en train de jouer et la kalista du joueur adverse, grâce à des prédicats "**dist**", "**bestDist**" et "**value**" Nous déterminons la distance minimale de toutes ces différentes distances via un prédicat "**min**".

- ➔ Si un pion du joueur adverse est plus proche de la kalista que le joueur est proche de la sienne, le prédicat permet de renvoyer un mouvement visant à essayer d'éliminer ce pion.
- ➔ Dans le cas contraire, on choisit le mouvement à faire en fonction des mouvements possibles (prédicat "**possibleMoves**") afin de se rapprocher au maximum de la kalista du joueur adverse.

Le prédicat général **'iaMove'** permet d'appeler les différents prédicats nécessaires à la recherche du meilleur mouvement à faire par l'ordinateur. L'appel au prédicat "**bestCoup**" permet de récupérer dans le prédicat "best", les coordonnées du mouvement choisi par l'ordinateur au terme de cette recherche. Le pion est ensuite déplacé (prédicat "**moveBestcoup**").

V. DIFFICULTES RENCONTREES ET AMELIORATIONS POSSIBLES

La prise en main du langage Prolog a été difficile au commencement du projet. En effet, ce langage ne ressemble à aucun langage de programmation que nous avons l'habitude de manipuler. La syntaxe, la manière de construire les prédicats récursifs et l'utilisation à bon escient du "cut" nous ont paru fastidieux au début du projet mais au fur et à mesure nous nous sommes habituées et cela nous a permis de réellement découvrir un nouveau type de raisonnement auquel nous n'avions pas été confrontées auparavant.

Nous avons également remarqué la difficulté de détecter d'où vient une erreur logique dans les différents prédicats.

La partie graphique du jeu a été difficile à mettre en place, il s'est avéré délicat de trouver une manière d'afficher un plateau de jeu facile à comprendre visuellement. En effet, nous avons commencé à coder le jeu et nous nous sommes intéressées à la partie graphique du jeu qu'à la fin du projet. Ainsi, nous pensions utiliser le même tableau pour gérer les coordonnées des pions que pour gérer la partie graphique mais nous nous sommes rendu compte que nous ne pouvions modifier la valeur des éléments des listes composant le plateau (par exemple pour le placement d'un pion, la valeur d'une case changerais de '1' à 'sR 1') car celle-ci était utilisée par plusieurs prédicats afin de donner au Khan la valeur correspondante à la case du dernier pion joué. Nous avons donc décidé de faire une partie graphique totalement indépendante des prédicats du jeu. Nous avons également essayé de trouver une manière de modéliser le fait que le Khan "coiffe" une pièce mais nous n'avons pas trouvé de solution adéquate, donc nous avons opté avec le fait d'afficher à la console sa valeur directement, en dessous du plateau de jeu à chaque tour.

La modélisation des différentes pièces du jeu nous a semblé complexe au premier abord. Dans le jeu de KHAN, il y a douze pièces, de deux types différents (sbire et Kalista) et de deux couleurs différentes (ocre et rouge). Nous avions ainsi pensé au départ à créer des prédicats propres aux sbires et à la kalista et pour chaque couleur. Nous avions donc par exemple les prédicats "sbireRouge1", "sbireOcre2", "kalistaRouge"… soit 12 prédicats différents pour les 12 pièces différentes. Mais plus nous avancions dans la conception de notre jeu, plus la modélisation des pions sous cette forme nous a posé des problèmes. En effet, pour chaque prédicat du jeu, il fallait créer plusieurs versions afin de prendre en compte les 12 différents prédicats des pièces, ce qui représentait énormément de "copier-coller" de code. Ainsi, il nous a paru beaucoup plus pratique de ne créer qu'un seul prédicat général "pion". Le type du pion est facilement identifiable grâce à la valeur de ses arguments, et le code en a été énormément simplifié.

Les vérifications systématiques des choix de l'utilisateur nous ont paru assez fastidieuses. Ainsi, le prédicat **"possibleMoves**" nous a semblé être le plus dur à implémenter. En raison des différentes règles présentes dans le jeu et de tous les mouvements possibles (le fait par exemple qu'un pion ne peut pas seulement se déplacer en ligne droite), il a fallu prendre en compte tous les cas possibles.

Concernant les améliorations, nous avons pensé que nous aurions pu initialiser les pièces de l'ordinateur avec une position stratégique seulement pour la partie "Humain vs Ordinateur".

Nous pensons également que l'IA de l'ordinateur aurait pu être plus complexe, par exemple en calculant pour chaque coup possible actuel, les différents coups suivants possibles pour le joueur adverse (algorithme minimax), et décider quel coup effectuer en fonction du résultat.

Notre code aurait également pu être plus simplifié et plus facile à comprendre d'un point de vue extérieur, en créant davantage de prédicat "parent" regroupant les différents prédicats "enfants".

VI. Conclusion

Ce projet fut une très bonne expérience. Malgré quelques difficultés au départ, le développement de ce projet a été formateur et nous a permis de mieux comprendre les principes du raisonnement algorithmique récursif. Ce projet nous a ainsi permis de mettre en pratique les concepts théoriques appris en cours, au niveau de la manipulation du langage prolog, mais surtout au niveau de l'implémentation d'une intelligence artificielle. La réflexion au niveau de la stratégie à adopter nous a paru la partie la plus complexe, mais également la plus intéressante de ce projet.