Siyu ZHANG
Mengjia SUI

IA02 - Projet : Jeu de KHAN

Création d'un système de jeu complet incluant une intelligence artificielle

Siyu ZHANG
Mengjia SUI

I. Introduction

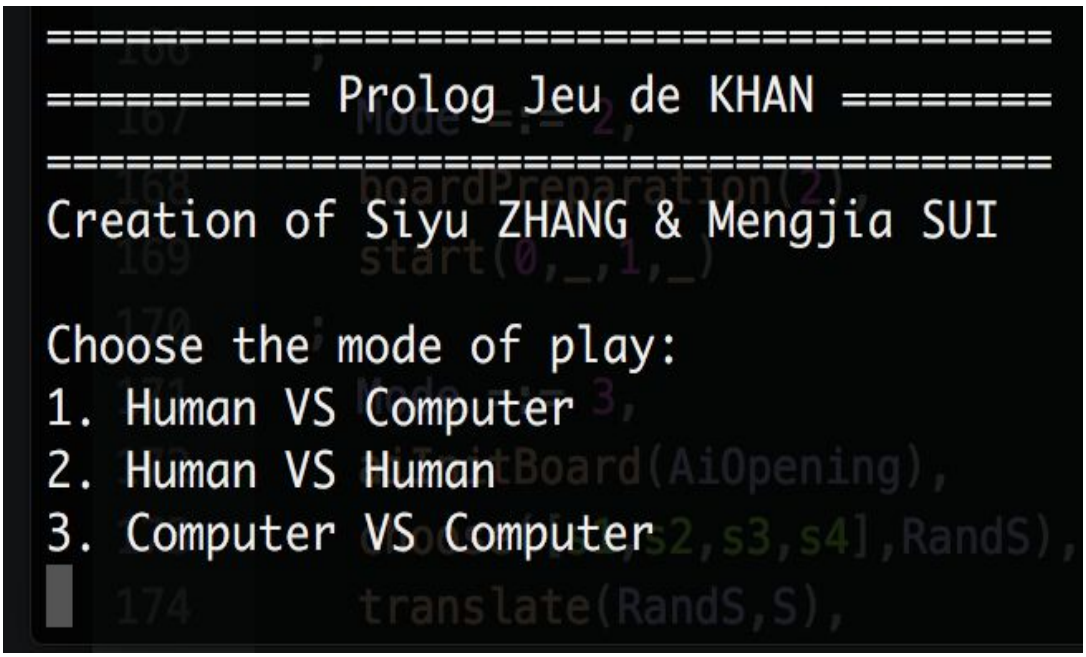
Le projet de l'UV IA02 pour ce semestre consiste a reproduire au moyen du langage Prolog un programme permettant de jouer au jeu du KHAN. Ce jeu devra permettre de jouer entres humains ou bien de défier l'intelligence artificielle qui est à développer avec le jeu.

Le jeu de KHAN est un jeu où deux joueurs tentent de pousser des pions sur un plateau de 6*6 cases. Malgré un plateau de taille réduite, la possibilité de coup est légion. Le but du jeu est de capturer la Reine adverse avec sa propre reine ou l'un de ses cinq serviteurs. Un pion du jeu peut se déplacer du nombre de cases correspondant au chiffre indique sur sa case de départ. Les déplacement sont verticaux et horizontaux avec possibilité de changer de direction en cours de route.

L'utilisateur du jeu peut faire fonctionner ce programme en trois mode différentes :

- Joueur contre Joueur : deux utilisateurs fassent à l'écran se passent les contrôles.
- Joueur contre Ordinateur : l'utilisateur joue contre le programme d'intelligence artificielle.
- Ordinateur contre Ordinateur : deux instances du programme s'affrontent, et le résultat est affiché à l'utilisateur.

Nous allons présenter l'organisation de notre programme, décrire les fonctionnements des prédicats principaux et les choix des structures. Nous poursuivrons en présentant l'intelligence artificielle. Enfin, nous présenterons les difficultés rencontrées au cours de ce projet et les améliorations possibles.



II. Organisation du programme

Notre programme Prolog est décomposé en cinq fichiers sources, contenant chacun une section différente des mécanismes du jeu. Nous ajoutons tout les fichier **movementRules.pl**, **jeuDeKHAN_lib.pl**, **translatedic.pl**, **opening.pl** dans le fichier principale **projet.pl.** Pour lancer le jeu, il faut appeler le prédicat *play/0* qui va alors présenter un menu de jeu permettant de commencer les différent modes de jeu.

Structure de données:

1. Plateau de jeu

La structure du plateau de jeu est la suivante:[TerrainMap,L,K]. C'est une liste contient tout d'abord l'information sur la terrain qui est une liste constante stocké dans terrainMap([2,3,1,2,2,3,2,1,3,1,3,1,1,3,2,3,1,2,3,1,2,1,3,2,2,3,1,3,1,3,2,1,3,2,2,1]), ensuite une liste de 12 éléments qui correspondant aux positions des 5 serviteurs et de la reine de chaque côté. K est une variable contenant la valeur de Khan, qui est initialement 0 et après entre 1 et 3 par rapport aux règles du jeu.

Cette structure est stocké en tant du fait dynamique avec le prédicat *board*/3.

- Exemple: *board* (TerrainMap ,[44,44,44,44,14,11, 44, 44, 44, 44, 1, 29], 1)

2. Coup

Un coup est une action effectuée par un joueur à un tour. Le coup est représente par une pair ainsi définie : [Origine, Destination] avec Origine la position du pion à déplacer et Destination la position à laquelle envoie on ce pion.

- Exemple: [1,31,32] (Pour déplacer le pion de côté1 de position *31* à position *32*)

jeuDeKHAN.pl

Dans ce premier fichier, on trouve les prédicats de base du jeu. Il s'agit principalement de fonctions suivantes:

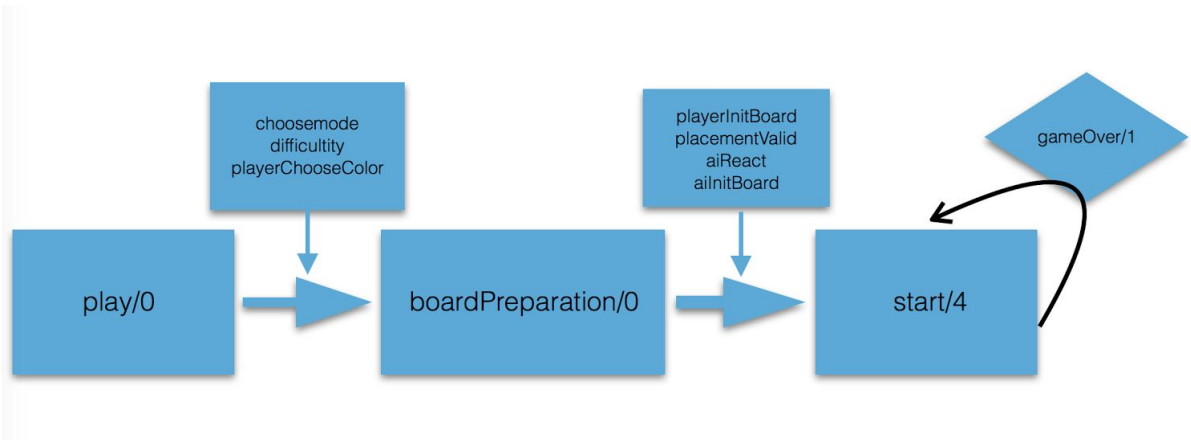


Figure 1 - Fonctionnement d'interface utilisateur (UI)

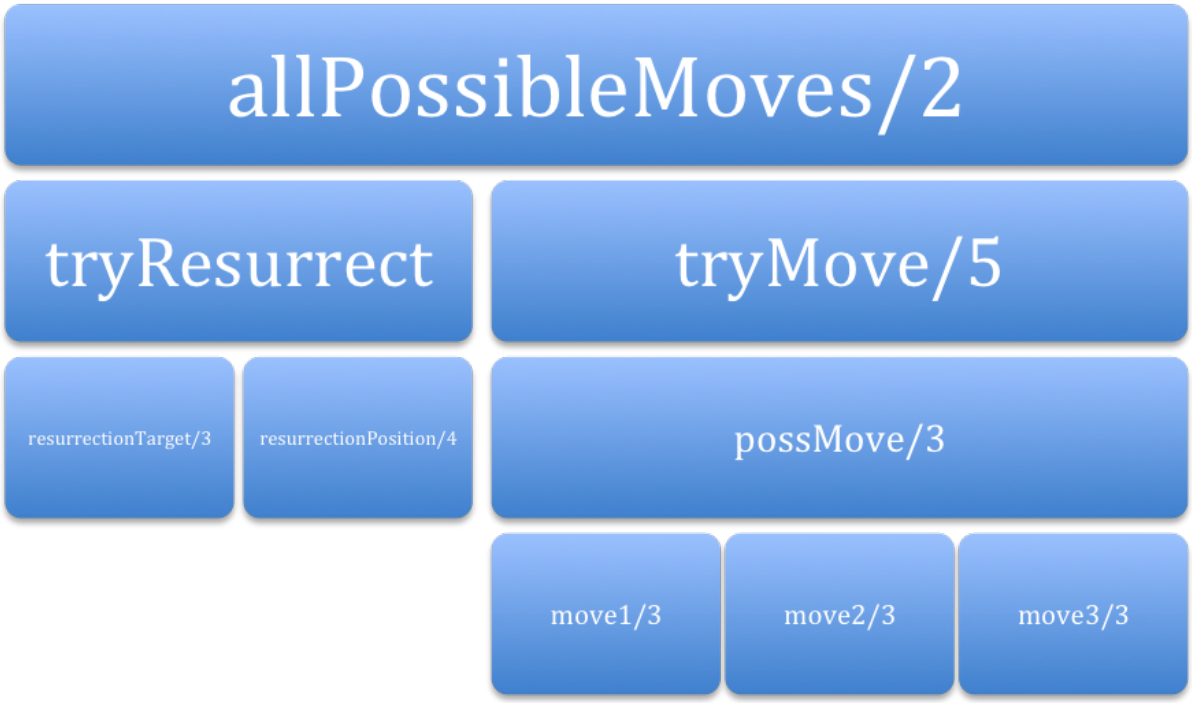


Figure 2 - Modélisation du mouvement

Explication précise sur des prédicats principaux:

L'utilisateur choisit le mode de jeux en entrant la valeur de mode, s'il veut jouer contre ordinateur ou ordinateur contre ordinateur, alors il va ensuite choisir le niveau puis le programme entrera dans le prédicat *boardPreparation*/1 correspondante.

initBoard: C'est un prédicat permettant de générer le plateau initial du jeu. Il généra le positionnement des pièces des différents joueurs. NB: Il est uniquement pour debug.
boardPreparation/1: Initialisation du plateau selon le cas: humain contre IA, pvp, IA contre IA.
movement: On distingue le premier déplacement des autres car il n'y a pas de khan avant le premier coup. Et le Khan est utilisé par l'intelligence artificielle pour choisir son prochain coup. Un mouvement entraîne la modification et la sauvegarde du *board/3*.

- Si le joueur c'est humain, il choisis un pion a déplacer et on appel le prédicat *tryMove*/5 pour tester si le pion qu'il veut déplacer bien respect les règles. Si c'est le cas, il va appeler le prédicat *allPossibleMove*/2 et si le déplacement que le joueur effectue est dans le liste des mouvements possible, alors le pion est déplacé, le valeur de Khan est changé et le *board* est modifié.
- Si le joueur c'est ordinateur, il se charge par la partie IA au dessous.

evaluate/4: A chaque tour de boucle du jeu, quand le plateau a été modifié, on examine état de reine de chaque côté , si une reine est disparu alors le jeu s'arrête, on entre dans la procédure *gameOver/1* et déterminer le gagnant.

Divers:

movementRules.pl	contient les règles basiques
jeuDeKHAN_lib.pl	contient les prédicats auxiliaires
translatedic.pl	une dictionnaire pour traduire les entrés d'utilisateur
opening.pl	aident l'IA à configurer le plateau au début du jeu

Siyu ZHANG
Mengjia SUI

III. Intelligence artificielle

Idée principale:

D'après le théorème de Zermelo, dans tous les jeu strictement compétitifs (les gains de l'un sont très exactement les pertes de l'autre) à deux joueurs à information parfaite (les positions des pièces, ainsi que leurs types sont connus par les deux joueurs), où les chances n'interviennent pas, existe une stratégie parfaite imbattable, soit pour le lanceur de jeu soit pour le défendeur. Mais pour trouver cette stratégie dans un jeu de grande échelle, il faut énorme de temps. (c'est la raison pour laquelle l'IA n'a pas toujours le gagnant dans le jeu d'échec / jeu de Go...)
https://en.wikipedia.org/wiki/Zermelo%27s_theorem_(game_theory)

Donc l'enjeu est de trouver un meilleur coup avec la ressource donnée et dans le temps donné. Pour ceci, on a utilisé l'algorithme de Mini-Max / Nega-Max.

Algorithme:

On fixe en amont un seuil de recherche, appelé la profondeur de la recherche, qui est lié à la difficulté choisie. La recherche se termine quand elle touche le seuil ou rencontre un état terminal du jeu (une des reines est prise), puis par une évaluation des jeux conduits, on détermine tel ou tel coup est favorable pour jouer comme le prochain coup. Donc une bonne fonction d'évaluation est essentielle pour qu'une IA soit vraiment intelligente. Une méthode alternative qui est l'algorithme de Monte-Carlo dépend moins la fonction d'évaluation, mais cette méthode est encore trop complique pour nous maintenant.

Notre fonction d'évaluation *evaluate/4* consiste deux partie: une note portée par les pièces, l'autre par leur mobilités. La première est quasiment évidente, on définit qu'un pion vaut 5 points et une reine vaut 1000 car la sécurité de la reine est suprême. La deuxième l'est beaucoup moins, d'apres notre experence dans une dixaine de jeu, on considere que la parite avec plus de pieces jouables a un avantage contre l'autre. Cet avantage est maximamisé quand une disobéissance de Khan a lieu. Donc on compte simplement le nombre possible du coup suivant. Afin de trouver un poids appropié avec lequel elle s'additionne avec la note des pieces, on reconstruit le jeu en Python.

Dans le code en Python, une procedure *AiVsAi* est utilisée pourqu'une IA joue contre une autre mais avec des parametre différents. On récolte le résultat après 100 jeux, et nous avons pu observer des résultats intéressants:

1) L'algorithme avec Alpha-Beta est équivalant à celui sans Alpha-Beta au niveau de qualité du coup généré. Ceci est deja prouvé dans 80s.

Pour économiser le temps de calcul, les expériences se font avec l'Alpha-Beta qui réduit le temps de réflexion à environs ½

Avec la notation ABn signifiant IA de profondeur n muni d'Alpha-Beta, on constate

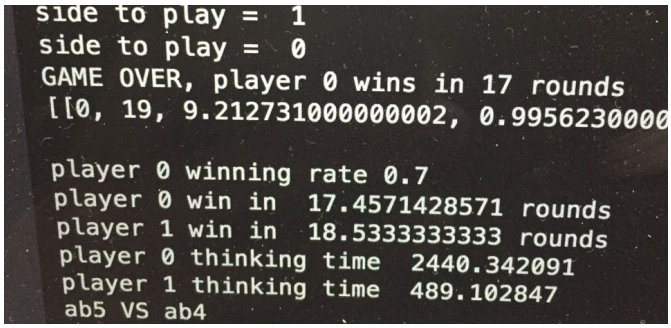
Sans évaluation de la mobilité:

2) AB3 vs No AI: taux de gain=98%, une IA aléatoire
3) AB4 vs AB3: taux de gain=69%
4) AB5 vs AB4: taux de gain=70%
5) AB5 vs AB3: taux de gain=76%, un résultat intéressant...
6) AB6 prend trop de temps au milieu du jeu

Avec un poids pour la mobilité(noté ABn poids), le temps de réflexion augmente de 200%, qui n'est pas en fait trop horrible comme il parait car de AB4->AB5, le temps monte 600%...

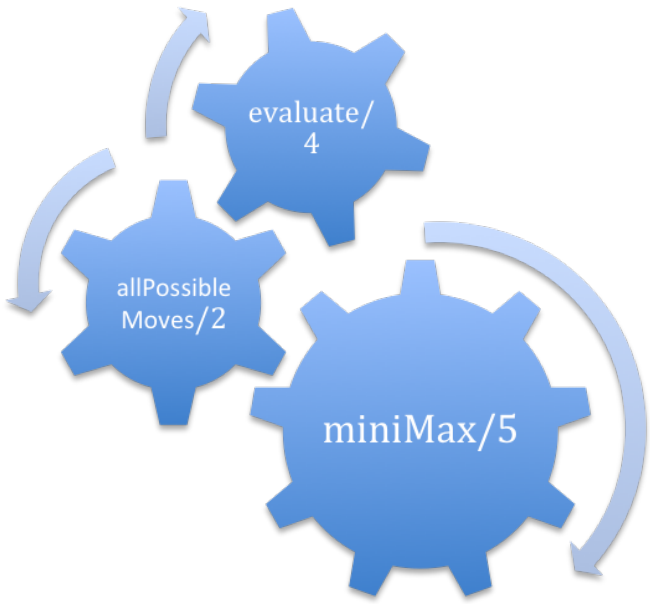
7) AB3 0,3 vs AB3: 61%
8) AB4 0,3 vs AB4: 63%

Ce poids est le meilleure qu'on a trouvé, on aurait dû apprendre et appliquer les techniques de Machine-learning comme la gradient descendante. Mais manque du temps à cause des autres UV... (et puis on ne dispose pas un ordinateur assez puissant pour faire ce genre de calcul qui dure jours et nuits)



Implémentation:

L'implémentation de Mini-Max est classique, elle ressemble beaucoup avec la version en Python. Faute de manquement de pouvoir retourner une valeur, une telle implémentation ne pourrait pas améliorer en équipant Alpha-Beta... Donc il vaut mieux de décomposer le gros prédicat en plusieurs petit prédicat...



Dans le fichier jeuAI.pl se trouvent les prédicats suivants:

1. *countPawns*/2 : Calcul de note matériale
2. *mobilityScore*/2 : Calcul de note de mobilité
3. *evaluate*/4 : Mixe les 2 notes
4. minimax

Début du jeu:

opening.pl

Pour répondre au besoin où le lanceur du jeu pourrait définit sa disposition, on définit un certain nombre de configurations possibles pour être prise au hasard par IA. Ces configurations proviennent de notre expérience du jeu.

En jouant le deuxième, IA a l'avantage de placer ces pièces vis-à-vis son adversaire, elle prend des configurations prédéfinies dans le *reactLib/1*, les applique Mini-Max, puis choisie celle avec la meilleure note.

IV. Prolog vs Python

Puisque la version Python est créée pour trouver la coefficient de la mobile, on a à plein implémenter la partie UI (mais cette version est quand même jouable), et le début du jeu est configuré au hasard.

En comparant ces deux versions, on peut constater la simplité de Prolog quand il s'agit de résoudre des probleme, ainsi que son inconvinnence par rapport à la modélisation et la manipulation des donnés. On peut donc dire que Prolog a sa propre élégance.

Pour la raison pédagogique, la notation des fonctions procédures et prédicats suivent le même logique.

V. difficultés rencontrées et améliorations possibles

Difficultés:

- logique, conflit avec le paradigme classique...

Manque de connaissance du Prolog au début, on a cherche sur Internet pendant la première TP pour affecter une variable globale à une liste...

Améliorations possibles:

- GUI

On aurait pu faire. Mais les motifs suivants on l'a abandonnée:

1) la GUI XPCE fonctionne que sur SWI-Prolog qui n'est pas installé sur l'ordinateur d'UTC.
2) XPCE n'est pas du tout du Prolog! (phrase extraite de son doc)

- Alpha-Beta, hash-table et histoire

Déjà expliquée dans la partie IA, et puis ce dernier on doute qu'il est même pas faisable dans Prolog.

- Auto-configuration par machine learning
- Une second réflexion sur la note de mobilité

D'abord cette note basé sur le nombre de possible du coup suivant prend un temps considérable à calculer.

Plus profondemment, avoir plus de choix à jouer n'est peut-être pas toujours un avantage, on cherche à obtenir une désobéissance du Khan pour soi-même, mais pour l'attendre, le nombre de choix doit descendre d'abord.

VI. Conclusion

Ce projet nous permet aborder les mécanismes de programmation logique avec logiciel Prolog. Nous avons crée un jeu en formalisant ses règles ainsi que de la manière de concevoir une intelligence artificielle en mesure de générer et de déterminer les coups.