PROJET IA02 – Le jeu KHAN



Rapport Final

I – Principaux prédicats du jeu et fonctionnement interne 3

I.1 – Les faits statiques 3

I.1.1 – Liste des faits statiques 3

I.1.2 – Utilisation des faits statiques 3

I.2 – Les faits dynamiques 3

III – L’affichage du plateau de jeu 4

IV – Placement des pions au tout début du jeu 4

V – Mouvements des pions au cours du jeu 4

V.1 - Prédicats de contrôle 4

V.2 - Prédicats de générations de coups et pions possibles à jouer 5

V.3 - Prédicats gérant le déplacement des pions 5

VI – Boucle de jeu et fin de la partie 6

I – Principaux prédicats du jeu et fonctionnement interne

I.1 – Les faits statiques

I.1.1 – Liste des faits statiques

Faits servant à définir les numéros des joueurs

* player1(1).
* player2(2).

Liste des pions

* pawnList(['S1', 'S2', 'S3', 'S4', 'S5', 'K'])

Plateau de jeu

* chooseBoardDisplay(Clé, PlateauVouluSelonClé)
* Fait statique qui permet au joueur de choisir la disposition du plateau qu’il veut

I.1.2 – Utilisation des faits statiques

Avoir la valeur d’une case ayant pour coordonnées X et Y

* get\_cell\_value(X, Y, CellValue).

Récupérer le code de l’autre joueur

* get\_other\_player(ActualPlayer, Player\_2)
* get\_other\_player(ActualPlayer, Player\_1)

Vérifier qu’une case de coordonnées X et Y soit sur le plateau

* cell\_in\_board(X,Y)

I.2 – Les faits dynamiques

Le plateau de jeu « actif »

* :-dynamic(activeBoard/1).
* Permet de sauvegarder et récupérer la disposition du plateau choisie par le joueur au début du jeu

Indices I et J qui servent au déplacement sur le plateau de jeu au cours de la partie

* :-dynamic(i/1).
* :-dynamic(j/1).

Prédicat pawn/4 qui sert à gérer les pions des joueurs sur le plateau

* :-dynamic(pawn/4).
* S’utilise de la façon suivante : pawn(Indice\_ligne, Indice\_colonne, Pion, Joueur)
* Permet énormément de choses :
  + En ayant la ligne et la colonne, on peut savoir s’il y a un pion sur la case et si oui, quel est ce pion et à quel joueur il appartient
  + En ayant le pion et le joueur, on peut savoir si le pion du joueur est sur le plateau et si oui, à quels indices il se situe
  + En ayant la ligne, la colonne et le joueur, on peut savoir si la case est vide, possède un pion adverse ou bien un pion du joueur.
* C’est un fait dynamique qui nous sera utile tout au long du jeu pour les déplacements des pions et le calcul des coups possibles
  + Permet de calculer la liste de tous les pions d’un joueur actuellement sur le plateau, de même que la liste des pions d’un joueur sortis du plateau

Prédicat khan/2 permettant de gérer la position du Khan

* :-dynamic(khan/2).
* Les deux arguments sont les indices de ligne et de colonne
* Fait dynamique qui est utile pour :
  + Récupérer la valeur de la case actuelle du Khan
  + Vérifier qu’une case possède la même valeur que la case actuelle du Khan

III – L’affichage du plateau de jeu

Affichage du plateau de jeu grâce au prédicat *dynamic\_display\_board(Board)* qui affiche la variable *Board* en utilisant les fait dynamiques *I* et *J*.

Nous avons aussi mis en place un un affichage des pions des joueurs en couleur afin de bien les distinguer entre eux.

IV – Placement des pions au tout début du jeu

Prédicats de contrôle servant à vérifier les informations entrées par l’utilisateur :

* Prédicat *pawn\_in\_initial\_lines(X,Y,Player)*
  + Vérifie, selon que *Player* soit le joueur 1 ou 2, que les coordonnées X et Y correspondent à une case située sur les deux premières lignes faisant face au joueur.
* Prédicat *initial\_pawn\_placement\_correct(Player, X, Y)*
  + Appelle le prédicat *pawn\_in\_initial\_lines(X,Y,Player)*
  + Vérifie que le joueur *Player* ne possède pas déjà un pion sur la case de coordonnées X et Y.

Placement initial des pions :

* Demander à chaque joueur de placer ses pions (on part de la liste des pions non utilisés des joueurs)
* A chaque itération, on demande à un joueur de placer un pion jusqu’à que la liste des pions inutilisés soit vide
  + Pour chaque placement, on vérifie bien que les coordonnées entrées par l’utilisateur soient correctes
* On place un pion grâce au prédicat *place\_pawn(X, Y, Pawn, Player)* qui fait appel au fait dynamique *pawn*.

V – Mouvements des pions au cours du jeu

V.1 - Prédicats de contrôle

*check\_good\_coordonates(X,Y,MoveList)*

* Prédicat qui vérifie que les coordonnées X et Y correspondent bien à des coordonnées présentes dans la liste MoveList représentant l’ensemble des coups possibles

*move\_right(X, Y, NY)*

* Prédicat qui vérifie qu’une pièce peut bouger vers la droite. Si c’est le cas, alors NY représente la valeur de Y après le mouvement
* Les prédicats suivants ont un fonctionnement similaire :  
  *move\_left(X, Y, NY)  
  move\_up(X, Y, NX)  
  move\_down(X, Y, NX)*

finish\_right(X, Y, NY, Player)

* Prédicat qui vérifie qu’une pièce peut bouger et terminer son mouvement vers la droite. Si c’est le cas, alors NY représente la valeur de Y après le mouvement. A la différence de *move\_right* qui vérifie si la case de droite est vide, il faut ici seulement vérifier que la case ne possède pas de pion du joueur. En effet, ce dernier ne peut pas manger ses propres pions mais peut manger ceux de l’adversaire.
* Les prédicats suivant ont un fonctionnement similaire :  
  *finish\_left(X, Y, NY, Player)  
  finish\_up(X, Y, NX, Player)*  
  *finish\_down(X, Y, NX, Player)*

V.2 - Prédicats de générations de coups et pions possibles à jouer

*get\_all\_cells\_according\_to\_khan\_cell\_value(CellList)*

* Prédicat qui va remplir la variable *CellList* avec l’ensemble des cases du plateau qui ont la même valeur que la case actuelle du Khan. De plus, toutes les cases contenues dans *CellList* à l’issue de l’appel seront aussi des cases vides.
* Ce prédicat est utilisé lorsqu’un joueur peut remettre un pion en jeu : il lui faut alors le remettre sur une case vide possédant la même valeur que la case actuellement occupée par la Khan.

*possible\_moves(X, Y, Player, Range, AlreadySeens, MoveList)*

* Prédicat qui permet de générer tous les coups possibles pour une case, un joueur et la distance en nombre de cases (1, 2 ou 3 selon la case de départ).
* La liste des coups possibles sera unifiée avec la variable *MoveList*.
* La variable *AlreadySeens* est là pour contenir l’ensemble des cases que l’on a déjà visité au cours de la recherche. En effet, un pion, lors d’un déplacement, ne peut pas revenir sur ses pas.
* La liste complète *MoveList* des cases de destination possibles en partant de la case X,Y et avec un déplacement d’une valeur de *Range* est obtenue en utilisant *setof* sur le prédicat *possible\_moves* :  
  *setof(ML, possible\_moves(I,J,Player,CellValue,[(I,J)],ML), ML),*  
  *flatten(ML, MoveList).*

*get\_possible\_pawn(Player, UsedPawnList, PossiblePawnList)*

* Prédicat qui retourne l’ensemble des pions qu’un joueur peut joueur en tenant compte de la position actuelle du Khan : un joueur est obligé de jouer une pièce se situant sur une case de même valeur que la case actuelle du Khan.
* Si le joueur n’a aucun pion sur une case de même valeur que celle du Khan, alors il pourra joueur l’ensemble de ses pions, y compris les pièces précédemment mangées par l’adversaire.

V.3 - Prédicats gérant le déplacement des pions

*ask\_pawn\_new\_position(I,J,MoveList,Pawn,Player)*

* Prédicat qui permet à un joueur de déplacer une pièce.
* La liste des coups possibles contenue dans *MoveList* est affichée
* Il lui est demandé de rentrer les nouvelles coordonnées
* Si les coordonnées sont mauvaises, on les lui redemande. Sinon, on bouge le pion en appelant le prédicat *move\_pawn(I, J, X, Y, Pawn, Player)*.  
  *I* et *J* sont les anciennes coordonnées, *X* et *Y* les nouvelles. *Pawn* représente le nom du pion à déplacer et *Player* contient la valeur (1 ou 2) du joueur.

*ask\_pawn\_new\_placement(MoveList,Pawn,Player)*

* Prédicat qui demande à un joueur de choisir la nouvelle position d'une pièce dans le cas où il remet la pièce sur le plateau alors qu'elle était sortie du jeu
* Fonctionnement identique au prédicat *ask\_pawn\_new\_position(I,J,MoveList,Pawn,Player)* à la différence qu’on a pas besoin des coordonnées d’origine et que *MoveList* contiendra la liste des cases obtenues via le prédicat *get\_all\_cells\_according\_to\_khan\_cell\_value(CellList)*.
* De plus, on place le pion non plus avec *move\_pawn* mais avec le prédicat *place\_pawn(X,Y,Pawn,Player)*

*ask\_movement\_to\_player(Player)*

* Principal prédicat permettant à un joueur donné (représenté par la variable *Player*) de choisir un pion à déplacer et sa destination.
* Utilisation du prédicat *get\_possible\_pawn* pour définir la liste des pions que le joueur peut jouer
* Si elle est vide, utiliser la liste des pions complète avec le fait statique *pawnList*.  
  Sinon garder la liste retournée par le prédicat *get\_possible\_pawn*.
* Demander quelle pièce veut jouer le joueur et vérifier que le joueur rentre bien le nom d’une pièce contenue dans la liste des pièces possibles à jouer.
* Si le joueur a choisi de replacer un pion, appeler le prédicat *ask\_pawn\_new\_placement* avec la liste des cases possibles obtenues via le prédicat *get\_all\_cells\_according\_to\_khan\_cell\_value*.
* Si je joueur a choisi de bouger un pion présent sur le plateau, appeler le prédicat *ask\_pawn\_new\_position* avec les liste des mouvements possibles obtenue via le *setof* sur le prédicat *possible\_moves*.

VI – Boucle de jeu et fin de la partie

La boucle de jeu permettant aux joueur de jouer l’un après l’autre jusqu’à que l’un deux prenne la Kalista adverse fait appel à trois prédicats.

*player\_win(Player)*

* Prédicat qui renvoi vrai si *Player* a gagné et faux sinon
* Il récupère la valeur du joueur adverse en utilisant le prédicat *get\_other\_player(Player,OtherPlayer)*
* Il construit la liste des pions non utilisés de *OtherPlayer*, c’est à dire les pions en dehors du plateau
* Et il teste enfin si la Kalista fait partie de cette liste de pions hors jeu

*human\_vs\_human\_game\_loop(Player)*

* Prédicat permettant de demander au joueur *Player* le mouvement voulu
* Il teste ensuite si ce joueur vient de gagner
* Si c’est le cas, le jeu s’arrête et le gagnant est félicité
* Sinon on rappelle récursivement *human\_vs\_human\_game\_loop* sur le joueur adverse

*human\_vs\_human\_launch\_game()*

* Prédicat permettant de lancer le jeu en appellant *human\_vs\_human\_game\_loop* sur le joueur 1.