

Projet IA02

Le jeu du SIAM



I) Fonctionnement des principaux prédicats

A) Lancement du jeu

Les prédicats principaux du jeu suivent les différentes phases du jeu. Ainsi, nous avons un premier prédicat afin de lancer le jeu.

Celui-ci propose à l'utilisateur de démarrer une partie Humain contre Humain, Humain contre Machine et Machine contre Machine. Il n'a besoin d'aucun argument.

Ces trois types de parties ont chacune leurs prédicats, afin de simuler le déroulement d'une partie.

- **partie_SIAM** permet de lancer une partie Humain contre Humain, qui commence par unifier un plateau initial, appelle un prédicat pour jouer un tour, et vérifie si on a atteint la condition de fin de partie (à savoir une montagne hors plateau), pour en afficher le gagnant.
- **partie_IA ...**
- **partie_SIAM_IA** permet de lancer une partie Machine contre Machine, où chaque IA jouera tour à tour.

B) Déroulement d'un tour de jeu

Un tour est découpé en deux parties, selon que le joueur est un Humain ou une IA.

- **tour** : saisie d'un coup, qui s'occupera de la vérification de la validité, puis prise en compte du coup, en « jouant », c'est-à-dire en modifiant le plateau. Un tour prend en argument un plateau, et renvoie un historique (vide s'il n'y a pas eu de poussée, instancié sinon), et un flag « Fin », qui permet de notifier au programme que la condition de fin de partie est remplie ou non.
- **tour_IA** : récupération de tous les coups possibles en fonction joueur, puis calcul du meilleur coup à jouer. Une fois le coup à jouer déterminé, on calcule l'historique destiné à être utilisé pour la modification du plateau (si poussée ou non), puis le programme « joue » le coup. Ce prédicat prend et renvoie les mêmes paramètres que le prédicat **tour**.

Lors d'une **saisie**, on demande à l'utilisateur de saisir un coup sous la forme (Départ, Arrivée, Orientation).

La case de départ est soumise à plusieurs vérifications :

- la validité tant syntaxique (integer)
- la validité de la case (case non vide et existante, pion appartenant au joueur).

Ces mêmes opérations sont effectuées sur les deux autres variables.

On vérifie ensuite que la case d'arrivée est valide. On peut donc ici tomber sur deux cas :

- La case est vide, on réalise le déplacement si la case existe
- La case est non vide, on vérifie alors que la poussée est possible via **poussee_possible**)

Détail du prédicat **poussee_possible** :

Le prédicat **poussee_possible** fait appel au prédicat **genere_liste_force_masse/5**

Le prédicat **genere_liste_force_masse/5 :**

But :

S'efface si la poussée est possible

Paramètres :

**genere_liste_force_masse(+Case, +Orientation, +Plateau, (?Force,?Masse),
-Historique)**

+Case :

Case sur laquelle le pion souhaite arriver

+Orientation :

Orientation du pion que l'on déplace

+Plateau :

Configuration du plateau de jeu complet

?Force :

Valeur de la force actuelle

?Masse :

Valeur de la masse actuelle

-Historique :

Retourne une liste des pions présents dans la file de poussée

Fonctionnement :

Introduction :

Le calcul de la file de poussée se fait en fonction de l'orientation du pion. On retrouve donc le prédicat **genere_liste_force_masse/5** avec les 4 valeurs pour le paramètre **+Orientation**.

Pour chacune de ces orientation, on retrouve 4 cas :

- La poussée d'une montagne
- La poussée d'un animal dans le meme sens que le pion
- La poussée d'un animal dans le sens opposé au pion
- La poussée d'un animal dont le sens n'impacte pas le poussée

L'arret du calcul de la file de poussée se fait grace à deux conditions d'arret :

- La case que l'on explore est vide
- La case que l'on explore est hors du plateau

Approfondissement et détails :

Prenons un exemple complet de calcul de file de poussée sur une poussée nord (n) de la case 51 vers la case 52.

55	R V				
54	R <				
53	M				
52	E ^				
51	E ^				

Trace d'exécution du prédicat **genere_liste_force_masse/5** :

Step 1 :

- 1) Appel du prédicat **genere_liste_force_masse(52, n, P, (1,0), H)** car un animal pousse avec une force de 1 et une masse de 0
- 2) Reconnait que la poussée se fait vers le nord :
 1. Test l'orientation du pion sur la case d'arrivée (52) et reconnaît que c'est la meme que celle du pion (51)
 2. Test si la poussée est valide et retourne ok

Step 2 :

- 1) Appel du prédicat **genere_liste_force_masse(53, n, P, (2,0), H)** car la force des animaux c'est additionnée
- 2) La poussée s'effectue toujours vers le nord :
 1. Test la case d'arrivée (53) et reconnaît une montagne
 2. Test si la poussée est valide et retourne ok

Step 3 :

- 1) Appel du prédicat **genere_liste_force_masse(54, n, P, (2,1), H)** car la masse de la montagne s'est ajoutée
- 2) La poussée s'effectue toujours vers le nord :
 1. Test l'orientation du pion sur la case d'arrivée (54) et reconnaît qu'elle n'est ni opposée ni identique à celle du pion (51)
 2. Test si la poussée est valide et retourne ok

Step 4 :

- 1) Appel du prédicat **genere_liste_force_masse(55, n, P, (2,1), H)** car la masse du pion précédent n'a rien modifié.
- 2) La poussée s'effectue toujours vers le nord :
 1. Test l'orientation du pion sur la case d'arrivée (55) et reconnaît qu'elle est opposée à celle du pion (51)
 2. Test si la poussée est valide et retourne ok

Step 5 :

- 1) Appel du prédicat **genere_liste_force_masse(56, n, P, (1,1), H)** car la force du pion précédent a réduit la force de poussée.
- 2) CONDITION D'ARRET CAR LA CASE 56 N'EST PAS UNE CASE VALIDE
- 3) Construction de l'historique à la remontée des prédicats

Une fois un coup déterminé comme valide, on renvoie le coup, sinon on demande à l'utilisateur de choisir un coup, tant que celui-ci n'est pas correct.

Pour une IA, on récupère tous les coups possibles de chaque pion grâce à **coups_possibles_joueurs**. Ce prédicat renvoie une liste, dont tous les éléments sont au même niveau (pas de liste dans d'autres listes), et prend en paramètre l'état actuel du plateau. Puis on fait appel au prédicat **meilleur_coup** afin de déterminer quel est le meilleur coup à jouer, en calculant une valeur pour chaque état à partir des états possibles du plateau.

Pour **jouer_coup**, qui s'occupe de modifier le plateau durablement grâce au coup déterminé au préalable, on commence par recopier le plateau actuel en changeant les bons pions grâce au coup et à l'historique. On peut ainsi supprimer le prédicat correspondant au plateau courant pour le réattribuer à une nouvelle valeur, ce qui permet, au début du tour suivant, de réunifier un nouveau plateau avec la nouvelle configuration.

Enfin, **fin_partie** prend en argument l'historique de poussée et le coup joué. On unifie une variable Plateau afin de travailler sur la plateau mis à jour et l'on cherche à savoir si une montagne est en dehors du plateau. Si c'est le cas, On utilise l'historique de poussée afin de déterminer le gagnant. On commence donc par inverser l'historique, afin de supprimer la montagne, et de vérifier dans l'ordre quel pion est le plus proche de la montagne et dans la bonne orientation. Si on arrive à un historique vide, c'est que la seule pièce étant dans la bonne orientation est celle qui a effectué la poussée, le joueur actuel. Sinon, on s'arrête au premier pion dans le bon sens, et on vérifie de quel type il est.

II) Implémentation des faits

Un coup est un triplet de variables : (*Départ, Arrivée, Orientation*).

- *Départ* correspond à la case où le pion que l'on veut jouer est situé.
- *Arrivée* à la case sur laquelle on veut déplacer le pion (elle est peut être vide ou non, selon que l'on essaie de pousser des pions ou pas)
- *Orientation* à l'orientation que prendra la pièce lors de son arrivée sur la case (celle-ci est restreinte en cas de poussée).

Un plateau est représenté par une liste divisée en quatre parties : [Elephants, Rhinoceros, Montagnes, Joueur].

Une telle structure permet de rendre plus efficace les prédicats en permettant de sélectionner directement l'élément de la liste qui nous intéresse, et rend les modifications du plateau à chaque tour plus aisées.

Une poussée est représentée grâce à un couple de deux variables, correspondant à une Force et une Masse. Pour chaque montagne rencontrée sur une ligne de poussée, on incrémente la Masse de 1. Pour chaque animal, on vérifie l'orientation de celui-ci, afin de déterminer les forces en présence. Si l'orientation est opposée à celle de la poussée, on décrémente la Force de 1. Si elle est identique, on incrémente la Force de 1. Si elle est quelconque, on ne le prend pas en compte.

III) Description de l'Intelligence Artificielle

- Les difficultés rencontrées et améliorations possibles
-

La mise en place de l'intelligence artificielle commence par la recherche de tous les coups possibles. Le prédicat **coups_possibles_joueur** permet de récupérer la liste de tous les coups envisageable en fonction d'un joueur (rhinocéros ou éléphant). Ce prédicat retourne dans une liste un ensemble de coups.

Cette liste de coups est transmise au prédicat **meilleur_coup**, qui en fonction de cette liste de coups retourne le coup permettant de converger vers la victoire, c'est-à-dire le coup qui va faire tomber la montagne du plateau.

Pour évaluer un coup donné, les prédicats **calculer_etat** et **comparer_etat** sont utilisés. Nous avons eu l'idée d'attribuer un score au coup passé en paramètre.

Nous avons également utilisé quelques heuristiques du type :

- un coup permettant de faire tomber une montagne se verra attribuer un score de 1000
- un coup permettant de pousser une montagne aura un score légèrement inférieur,
- etc.

Ces prédicats entraînent le calcul des distances d'un animal à une montagne. Ainsi, les scores des coups sont comparés deux à deux, et l'on garde le coup qui a obtenu le meilleur score.

IV) Bilan du projet

Ce projet nous à permis d'apprendre à exploiter un moteur d'interprétation prolog en implémentant des prédicats de manipulation de liste assez complexes.

Il nous à également permis d'utiliser des prédicats déjà implémentés au sein du moteur.

Nous avons également compris comment utiliser la base de fait dynamique en faisant évoluer l'état du plateau au fur et à mesure de l'avancement de la partie

Cependant, nous avons manqué de temps pour réaliser une intelligence artificielle optimisée et réellement « intelligente ». Nous avons cependant pris conscience de la difficulté à anticiper les coups sur un jeu de plateau comme le Siam. Nous avons pu voir à travers certaines exécutions de notre programme, que le nombre de coups possibles étaient assez colossaux.

Nous aurions aimé avoir plus d'indication sur la mise en place de l'intelligence artificielle en étant guidé dans l'implémentation d'un algorithme de la théorie des jeux comme par exemple le minimax alpha beta.