REANEY Olivia GI02
DAMBRINE Florian GI02

utc
Université de Technologie
Compiègne

Projet IA02
Le jeu du SIAM

I) Fonctionnement des principaux prédicats

A) Lancement du jeu

Les prédicats principaux du jeux 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_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 où 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 vérification :

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

Ces mêmes opération 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

Paramétres :

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 paramétre *+Orientation*.

Pour chacune de ces orientation, on retrouve 4 cas :

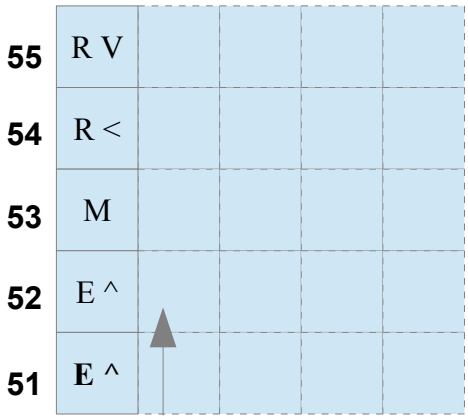
- La poussée d'une montagne
- La poussée d'un animal dans le meme sens que le pion
- La poussée d'un animal dans le sens opposé au pion
- La poussée d'un animal dont le sens n'impacte pas le poussée

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.



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 chercher à savoir su 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

Notre intelligence artificielle est très basique. Elle ne prévoit pas plusieurs coups à l'avance, mais calcule une valeur pour chaque état possible à partir de la liste de coups passées en argument. Afin d'instancier cette valeur, elle additionne un chiffre issu de la position des montagnes sur le plateau, un venant des positions des pions du joueur par rapport aux montagnes, et retranche un chiffre issu des positions des pions de l'adversaire. Cela permet ainsi de vérifier qu'un coup, qui nous permettrait d'avancer vers les montagnes, ne serait pas également bénéfique (voire plus) pour notre adversaire.

Le calcul de la position des montagnes se fait simplement : on additionne chaque valeur de chaque montagne. Ainsi, une montagne hors plateau sera valuée par 1000, une montagne sur les cases extérieures par 500, une montagne sur les cases intermédiaires par 250 et la case centrale par 125. Une telle différence entre les valeurs est nécessaire pour être à peu près sûr que ce sera toujours le coup qui entraîne une sortie de montagne qui sera valorisée.

De même, on calcule un « pourcentage » de distance pour les pions par rapport à toutes les montagnes. Ainsi, on commence par évaluer la distance entre une montagne et un pion, en faisant la différence entre la case d'une montagne et celle du pion. Comme on estime que cette valeur est une distance, on prend toujours la valeur absolue. On divise ensuite cette valeur par 10, et on prend l'inverse, ce qui inverse l'ordre des valeurs, et on multiplie de nouveau ce chiffre par 100. Cela permet donc d'avoir, lorsque les pions sont les plus éloignés, des valeurs de 36 ou 44 (et donc de 100/3,6 = 27,77 et 100/4,4 = 22,72) et lorsque les pions sont les plus proches, des valeurs de 10 ou 1 (et donc de 100/1 = 100 et 100/0,1 = 1000).

Après avoir additionné toutes ces valeurs, on vérifie pour chaque coup si la valeur du plateau possible est plus élevée ou non que celle d'avant. Si elle est plus élevée, le coup associé devient le nouveau coup à jouer, sinon on garde le même coup. On continue à vérifier ainsi jusqu'à ce que tous les coups soient épuisés.

IV) Bilan du projet

Ce projet nous a permis de nous rendre compte de la puissance du langage Prolog pour programmer des algorithmes avec des contraintes claires et prédéfinies (comme c'est le cas pour des règles de jeux), mais également la facilité de stocker des données dynamiquement au cours du programme. Cependant, il est clair que sa syntaxe et son fonctionnement sont loin des langages vus préalablement, il nous a parfois été difficile de mettre en place les fonctions sans les reprendre plusieurs fois (notamment lorsque nous avions besoin d'itérer sur des éléments) lorsqu'on a tendance à penser dans d'autres langages mieux connus. Cela permet aussi de faire travailler la réflexion sur la récursivité et la portée des variables, surtout dans les cas où nous avons du utiliser des semblants de boucles avec le prédicat **repeat**.

Malgré le fait que nous ayons mis en place une intelligence artificielle, assez sommaire, nous aurions pu aller plus loin en essayant de prévoir plusieurs coups à l'avance. Pour cela, il aurait fallu calculer les états possibles suivant les états déjà calculés, et ce à un n-degré de profondeur. Mais le Prolog est un langage assez complexe, et calculer n-degrés aurait vite demandé un temps de calcul relativement long. Nous avons essayé de nous prémunir des erreurs en vérifiant la syntaxe des données rentrées par l'utilisateur, soit grâce à l'utilisation de prédicats prédéfinies, tels que **number**, **atom**, mais également grâce à l'utilisation d'options dans le prédicat **read_term**, où on précise qu'une levée d'exception ne renverra plus une erreur de syntaxe mais entraînera simplement l'évaluation du prédicat à fail, ce qui permet à l'utilisateur de rentrer une variable sans avoir à relancer le jeu. Nous avons également défini des prédicats pour vérifier plus simplement des valeurs (par exemple un déplacement n'est possible que si le symbole passé dans le prédicat **deplacement_possible** s'unifie avec 10, -10, 1 ou -1). Nous avons cherché à utiliser des expressions régulières, mais leur utilisation nous paraissait compliquée par rapport à la simplicité des valeurs voulues.

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.