P12

Projet IA02

Jeu de SIAM

Implémentation en Prolog du jeu de plateau SIAM

VANDECNOCKE Yoann – LANCELOT Simon
IA02
P12

Introduction

Pour ce projet, nous devions développer une implémentation du jeu de plateau « SIAM » en Prolog.

Pour ce faire, le projet était composé de trois parties : affichage du plateau, jeu humain, intelligence artificielle.

Le jeu consiste en un affrontement entre des éléphants et des rhinocéros, sachant que la partie s'arrête lorsqu'une montagne sort du plateau.

Il y a au maximum 5 éléphants et 5 rhinocéros sur le plateau, et 3 montagnes initialement.

Affichage du plateau

La première partie de ce projet consistait donc en l'affichage du plateau de jeu. Cette partie sera présente tout au long du jeu, car à chaque coup joué, ainsi qu'à l'ouverture du jeu, le joueur (si c'est un humain), doit pouvoir visualiser l'état du plateau, sans quoi il ne pourrait pas jouer.

A l'inverse, l'affichage du plateau n'est pas utile pour l'intelligence artificielle.

Affichage classique

Le principe a tout d'abord été d'afficher une matrice 5*5 avec les numéros des cases. Pour cela, nous avons utilisé une « boucle » imbriquée. En effet, pour chaque ligne de la matrice, nous affichons le bon nombre de cases.

C'est donc le prédicat *affiche_lignes*, qui fera appel au prédicat *affiche_case*. Ici, pour chaque ligne, on aura 5 itérations de *affiche_case*, il y aura donc 5 cases affichées par ligne. *Affiche_lignes* fera également 5 itérations, ce qui permettra d'afficher la matrice 5*5 complète.

Puisque l'on veut que les numéros de la matrice soient organisés de la sorte : numéro_ligne-numéro_colonne, le numéro de chaque case sera alors **K = I*10+J**.

A l'issue de cette étape, nous avions donc la matrice avec les numéros de cases affichés.

Affichage des éléphants, des rhinocéros et des montagnes

La deuxième étape consistait à afficher les éléphants, rhinocéros et montagnes sur le plateau de jeu. Pour cela, nous avons trouvé utile de créer trois prédicats importants : *cherche_E, cherche_R* et *cherche_MI*.

Ces prédicats fonctionnent de la même façon, à la différence qu'ils ne cherchent pas la même chose évidemment. On se place respectivement dans la partie que l'on souhaite du plateau (la première partie pour *cherche_E*, la deuxième pour *cherche_R* et la troisième pour *cherche_M)*.

Ensuite, grâce au prédicat **member**, on parcourt les sous listes d'éléphants, de rhinocéros, ou la liste de montagnes, et on vérifie si le numéro de la case dont l'on souhaite connaître le contenu en fait partie.

Si c'est le cas, alors au lieu d'afficher le numéro de la case, on affichera **E**, **R** ou **/**. De plus, on envoie ensuite un prédicat *verif_or* qui permet d'afficher l'orientation des animaux.

Enfin, on a un prédicat *plateau_initial* et un prédicat *affiche_plateau*. Le premier initialise la partie avec un premier plateau, et le second permet d'appeler les prédicats définis ci-dessus à partir d'un plateau passé en paramètre.

Jeu humain

Cette deuxième partie consiste à implémenter le jeu pour que des humains puissent disputer une partie l'un contre l'autre.

Le prédicat *plateau_depart* demandé est en fait *plateau_initial* développé dans la partie précédente.

Cette partie du projet est surement la plus importante, car c'est elle qui va permettre le déroulement d'une partie.

Le prédicat *coup_possible*

Il nous était demandé de développer plusieurs prédicats, notamment *coup_possible*, qui, à partir d'un coup, devait déterminer s'il était possible de le jouer.

Il semble nécessaire de définir ce que l'on entend par « coup ». Un coup dans ce jeu est en fait une combinaison d'une case de départ, d'une case d'arrivée et d'une orientation. Nous avons choisi de ne

pas entrer un coup comme un triplet (Départ, Arrivée, Orientation), mais comme trois paramètres distincts.

Cependant, avant de définir ce prédicat, il est obligatoire de se renseigner dans les règles du jeu afin de voir quels sont effectivement les coups possibles, et d'implémenter tout d'abord ces différents coups sous forme logique.

D'après les règles, les coups possibles sont : déplacement d'une case, changement d'orientation en restant sur la même case, poussée.

Les deux premiers coups ne nous ont pas posé trop de problèmes, au contraire de la **poussée**.

Déplacement possible

Pour ce prédicat, nous avons considéré le déplacement vers une case libre du plateau, mais aussi l'entrée d'une pièce sur le plateau, et, à l'inverse, la sortie d'une pièce du plateau.

Une pièce peut se déplacer dans les 4 directions et ne peut se déplacer que d'une case. Ainsi, on va appeler *déplacement_possible* avec P (plateau), D (case départ) et A (case arrivée). On va tout d'abord vérifier si la case d'arrivée voulue est bien libre, grâce au prédicat *case_libre*, et ensuite vérifier dans quelle direction est le déplacement.

Pour cela, la règle est simple, si on se déplace vers l'est, la case d'arrivée doit être **case_départ+1**, si on se déplace vers l'ouest, **case_arrivée = case_départ-1**, si on va vers le nord, **case_arrivée = case_départ+10** et si on va vers le sud, **case_arrivée = case_départ-10**.

Il y a deux cas où il faut faire attention, pour l'entrée et pour la sortie d'une pièce du plateau. Dans le cas où on va entrer une pièce, la case de départ sera 0. Il faut donc vérifier que la case d'arrivée est une case extérieure du plateau, grâce au prédicat *verif_case_ext* et que la case d'arrivée est libre.

Pour la sortie d'une pièce, il suffit de regarder que la case de départ soit bien une case extérieure, la case d'arrivée est 0 dans le cas de la sortie.

Le prédicat *case_libre* est simple. Ce prédicat sera vrai s'il n'y a ni éléphant, ni rhinocéros, ni montagne sur la case. Il suffit d'utiliser les négations des prédicats utilisés précédemment.

Pour vérifier qu'une case est bien extérieure, le prédicat *verif_case_ext* fait plusieurs calculs afin de vérifier que l'on se situe bien sur les bonnes cases. En effet, par exemple, toutes les cases se terminant par **5**, donc [15, 25, 35, 45, 55] sont des cases extérieures. Ce sont également les seules cases divisibles par 5. Donc on sait que si une case est divisible par 5, alors ce sera une case extérieure.

Nous aurions pu vérifier de manière brutale, si la case faisait partie de la liste [15, 25, 35, 45, 55] par exemple, mais cela signifie que dans le cas où l'on veut réutiliser ce prédicat pour une autre matrice, cela ne fonctionnerait pas.

Changement d'orientation

Pour le changement d'orientation, nous n'avons pas créé de prédicat à proprement parler, mais à l'intérieur de *coup_possible*, nous testerons si la case d'arrivée est identique à la case de départ. Si c'est le cas, le seul coup possible est le changement de direction.

Il suffit alors de récupérer la direction initiale de la pièce, grâce à *get_orientation* (expliqué un peu plus loin), et de vérifier que cette direction est différente de celle passée en paramètre (sans quoi le coup est inutile, et n'en est même pas un).

Poussée possible

C'est la partie de *coup_possible* qui nous a posé le plus de problèmes.

Comme cela nous a été expliqué en TP, le principe était de retourner une liste contenant l'orientation des pièces se trouvant dans le sens de la poussée, et « m » quand il s'agissait d'une montagne.

Ensuite, nous avions deux variables **F** et **Masse**, qui sont respectivement la force de poussée des animaux et la masse des montagnes, et tant que F est supérieur à 0 et est supérieur ou égal à Masse, alors la poussée est possible.

Le plus compliqué pour nous n'a pas été de vérifier ces conditions, mais a été de renvoyer la liste que nous souhaitions, afin de la passer en paramètre d'un autre prédicat qui allait évaluer la possibilité du coup.

La vérification des conditions est effectuée dans le prédicat *poussee_possible*, où l'on vérifie à chaque fois que la force et la masse remplissent les bonnes conditions.

La récupération de la liste se fait, elle, dans le prédicat *recup*. Ce prédicat prend en paramètres un plateau, une case, une direction, et la liste vide. On va tout d'abord vérifier s'il y a un successeur, à l'aide de *successeur* (expliqué ci-dessous). La subtilité est que le successeur va être demandé dans la direction de la poussée initiale, et non pas forcément dans la direction de la case qui appelle le prédicat. Pour chaque successeur, on va alors vérifier son orientation, grâce à *get_orientation*, puis on va concaténer dans une liste (à la première itération, on va concaténer avec la liste vide) toutes les orientations des successeurs.

Une fois cette liste récupérée, on crée une nouvelle liste qui sera composée de cette liste, précédée de la direction de la poussée initiale, puis on enverra le prédicat *poussee_possible* avec cette liste, et une force initiale de 1, car dans tous les cas la force sera de 1 avec la première pièce, mais une masse de 0.

Nous avons parlé plus haut du prédicat *get_orientation*. Ce prédicat est défini simplement, il prend tout le plateau en paramètre, ainsi qu'une case, et retourne l'orientation de cette case. Pour cela, on

teste si la case fait partie des éléphants, auquel cas on renvoie son orientation, ou des rhinocéros, avec le même résultat. Si elle fait partie des montagnes, on force le prédicat à retourner « m ».

Le prédicat *successeur* prend en paramètres un plateau, une case de départ et une orientation, et retourne le numéro de la case d'arrivée. Pour chaque orientation, on va tester de quelle catégorie fait partie le successeur de la case. Ensuite, selon l'orientation, A prendra une certaine valeur.

Par exemple, si l'orientation est l'est et que la case dont on veut connaître le successeur est 11, alors on va tester si 12 fait partie des éléphants, des rhinocéros ou des montagnes. Si c'est le cas, on retournera alors 12.

Le prédicat *jouer_coup*

Pour ce prédicat, qui prend en paramètre un plateau initial, un coup et qui retourne un nouveau plateau, nous avons isolé plusieurs cas.

Tout d'abord, il y a deux types de coup, parmi les coups disponibles : ceux dont la case d'arrivée est identique à la case de départ, et les autres.

Ensuite, nous avons inclus de la robustesse dans notre programme, pour les cas où l'utilisateur entre un coup qui n'est pas possible, on vérifie donc à chaque fois *non coup_possible*, et dans le cas où le coup n'est pas possible, alors le nouveau plateau est égal au plateau initial, et c'est au même joueur de jouer.

Enfin, pour les cas où la case de départ est différente de la case d'arrivée, il faut différencier encore deux cas : la case d'arrivée est libre, et alors on se trouve dans le cas d'un déplacement simple, ou la case d'arrivée n'est pas libre et on se trouve dans le cas d'une poussée.

Si la case d'arrivée est libre, alors on regarde si le coup est possible, puis on appelle le prédicat *modif_plateau* (expliqué ci-dessous).

Si la case d'arrivée est occupée, et que la poussée est possible, plusieurs cas sont envisagés. Soit le coup est gagnant, soit il ne l'est pas. On vérifie donc après chaque poussée si le jeu continu ou si on s'arrête à l'aide du prédicat *continuer* qui est faux si une montagne est sortie du plateau. Si le jeu continu, on appelle le prédicat *pousse_plateau,* très similaire à *modif_plateau,* en lui passant en paramètre la liste des acteurs de la poussée. Si le jeu s'arrête, il y a un gagnant. Le prédicat *qui_gagne* affiche alors le vainqueur.

Le prédicat *modif_plateau*

Ce prédicat permet donc de modifier un plateau, passé en paramètre avec un coup et qui retourne un nouveau plateau. On différencie tout d'abord le cas des éléphants du cas des rhinocéros. Si la case de départ fait partie de l'une ou l'autre des catégories animales, alors on retire l'animal de

la case de départ, en remplaçant cette case par le couple **(0,0)**, dans la liste, puis on ajoute l'animal dans une case libre, en remplaçant cette fois le premier couple **(0,0)** que l'on trouve par le couple **(case, orientation)** de l'animal.

Ensuite, il suffit d'appliquer ces changements au plateau P2, qui sera le nouveau plateau.

La récursivité se fait au niveau de la catégorie. En effet, on regarde par exemple si (11,e) est la tête des éléphants, et si ce n'est pas le cas, on parcourt toute la liste des éléphants. Si ce n'est toujours pas le cas, alors on passe aux rhinocéros.

Le prédicat *pousse_plateau*

Ce prédicat est sensiblement le même que *modif_plateau* à la différence près qu'il décale une à une les pièces intervenues dans la poussée, en commençant par la dernière.

Le prédicat *qui_gagne*

Ce prédicat est appelé lorsqu'une montagne est sortie du plateau. Elle prend en paramètre la liste des acteurs de la poussée, le premier élément de la liste étant la montagne elle-même, le deuxième, la pièce juste derrière celle-ci, et ainsi de suite. On va regarder ici si la pièce la plus proche de la montagne est dans la même direction de la poussée, sinon, on regarde la case d'après. La boucle s'arrête dès qu'une pièce est dans la même direction, et renvoie donc comme vainqueur le joueur possédant cette pièce.

Boucle de jeu

Pour jouer, on commence tout d'abord par le prédicat *initialisation*, qui, comme son nom l'indique, va appeler *plateau_initial*, ce qui va donc retourner le plateau de départ, puis on va appeler *jouer* avec ce plateau.

Dans *jouer*, on va tout d'abord afficher le plateau passé en paramètre (donc le plateau initial lors de la première itération). Ensuite, on va demander à l'utilisateur de saisir une case de départ, une case d'arrivée ainsi qu'une orientation.

En fonction de cela, on va appeler le prédicat *jouer_coup*, puis à la sortie de ce prédicat, on va relancer *jouer* avec le nouveau plateau, qui est retourné par *jouer_coup*.

Ainsi, le nouveau plateau devient le plateau initial, et on recommence la boucle, jusqu'à ce qu'il y ait un gagnant.

Intelligence artificielle

Nous avons choisi de manière arbitraire que lors d'un jeu humain contre intelligence artificielle, l'humain jouerait avec les rhinocéros et l'IA avec les éléphants.

Le prédicat *coups_possibles*

Ce prédicat prend, pour notre part, trois paramètres : un plateau de jeu, une liste temporaire et une liste qui retournera la liste de coups possibles. Chaque coup de cette liste sera de la forme **(Départ, Arrivée, Orientation)**.

Nous avons défini trois cas de figure (il manque le cas où la pièce sort du plateau). Ces trois cas sont : lorsque la pièce de départ va pousser celle d'arrivée, lorsque l'on veut entrer une pièce sur le plateau, et enfin lorsque l'on veut se déplacer sur une case libre.

Etant donné notre définition du prédicat *successeur*, on peut dire qu'une pièce qui veut se déplacer sur une case libre n'a pas de successeur. Ainsi, si c'est le cas, on va, grâce à un **setof**, récupérer les cases disponibles pour se déplacer dans les 4 directions. Pour cela, on fait appel à *succ_libre*, prédicat créé uniquement pour cette partie, et qui va retourner une case d'arrivée à partir d'une case de départ.

Exemple : si l'on envoie *succ_libre* avec comme case de départ 11, il va retourner, avec l'aide de **setof**, les cases 21 et 12, si elles ne sont pas occupées par des pièces.

Le deuxième cas correspond au cas de la poussée, si la case de départ retourne un successeur. Dans ce cas, on va utiliser un setof également avec le prédicat *coup_possible*, au singulier, afin de récupérer tous les coups disponibles pour la poussée.

Le dernier cas correspond à l'entrée sur un plateau, donc quand on a un couple **(0,0)**, cela signifie que l'on peut entrer une pièce sur le plateau.

Dans ce cas, on va récupérer toutes les cases extérieures disponibles, grâce au nouveau prédicat *verif_case_exterieur*. Ce prédicat n'est pas codé très « proprement » mais permet de retourner ce que l'on souhaite.

Le problème est que dans ce cas, l'orientation est 0. Nous n'avons pas trouvé d'autre moyen que de la changer manuellement, dans les prédicats *jouer_ia* et *jouer_ia_humain*, et de définir arbitrairement que lorsque l'IA rentre une pièce sur le plateau, son orientation sera toujours à l'est.

Evidemment, cela n'est pas optimal, car il faudrait pouvoir afficher TOUS les coups, par exemple (0, 55, e), (0, 55, o), (0, 55, n) et (0, 55, s) si l'on entre une pièce en case 55.

Une fois la liste des coups possibles récupérée, pour un éléphant, on concatène afin de pouvoir récupérer dans une même liste les coups possibles de tous les éléphants.

Le prédicat *meilleur_coup*

Afin de développer une IA qui, au moins, pouvait effectuer d'elle-même quelques coups, nous avons choisi arbitrairement que l'IA jouerait toujours le premier coup disponible. Ce n'est clairement pas la bonne solution, mais nous ne sommes pas parvenus à faire jouer l'IA avec des méthodes plus complexes.

En effet, pour développer une IA qui soit vraiment intéressante, il aurait fallu que nous trouvions le moyen de récupérer le coup le plus proche de l'état final, c'est-à-dire quand une montagne sort du jeu.

Ainsi, dans notre cas, lorsque l'on fait jouer l'IA, elle va récupérer les coups possibles dans le plateau actuel du jeu, et envoyer le prédicat *jouer_coup* avec le premier coup disponible. Ce prédicat va effectuer un changement de plateau.

Lors du jeu IA VS IA, on va alors renvoyer *jouer_ia* avec le nouveau plateau et la liste de coups possibles correspondant à ce plateau.

Lors du jeu IA VS Humain, on ne va jouer qu'un coup avec l'IA, puis on va appeler une fonction de jeu classique de l'humain, sans boucle cette fois, et ce prédicat de jeu humain appellera lui-même à nouveau le jeu de l'IA.

Nous n'avons malheureusement pas réussi à mettre en place correctement une IA qui fonctionne bien. En effet, elle va jouer des coups jusqu'à ce qu'elle ne puisse plus entrer d'éléphants sur le plateau, et ensuite elle ne va pas continuer.

Difficultés rencontrées et améliorations possibles

Difficultés rencontrées

Au cours de ce projet, nous avons rencontré nombre de difficultés, dans chaque partie. La partie qui nous a pris le plus de temps et d'investissement étant sans conteste le jeu humain.

Au niveau de l'affichage, les difficultés n'étaient pas très importantes, ce qui nous a posé problème était de conserver le plateau et de le transmettre en paramètre de prédicat en prédicat, afin de pouvoir le modifier et d'en retourner un nouveau plus tard.

Pour le jeu humain, il y a eu plusieurs difficultés. Les deux plus importantes concernent la poussée possible et la modification du plateau en cas de poussée.

Pour la poussée possible, nous avons utilisé l'algorithme qui nous a été donné en TP, mais ce qui nous a pris du temps est le prédicat *recup*, qui permet de retourner la liste que l'on va utiliser pour

calculer la force et la masse. En effet, nous avions placé l'appel récursif au mauvais endroit, ce qui entraînait un backtracking, sui finissait par retourner la liste vide.

Pour la modification du plateau en cas de poussée, nous n'arrivions pas à faire bouger plusieurs pièces sur le même coup, l'idée initiale étant de bouger en premier la dernière pièce, afin de libérer une case libre pour la pièce d'avant, et ainsi de suite. Nous avons cependant réglé ce problème.

Améliorations possibles

Les grosses améliorations et difficultés concernent l'intelligence artificielle. En effet, nous avons remarqué plusieurs erreurs dans le prédicat *coups_possibles*. Ce prédicat ne prend pas en compte tous les coups possibles, et de plus renvoie certaines fois des coups impossibles, car si par exemple il a déjà traité le cas d'une pièce située en 13, et qu'après il veut entrer une pièce sur le plateau, alors il va essayer de la rentrer en 13 quand même.

De plus, il s'arrête lorsque tous les éléphants sont sur le plateau, sans les bouger.

Une autre amélioration concerne la robustesse du programme. En effet, il est robuste en cas de coup impossible mais il ne tient pas compte des erreurs de saisie.