Chicago Stock Exchange

Projet Prolog - IA02

Benquet Florent/ Talouka Oumaima - 16 juin 2015

Introduction

Dans le cadre du programme portant sur la programmation Logique en IA02, il nous a été demandé d'implémenter en Prolog le jeu « Chicago Stock Exchange » complet composé de la partie Humain vs Humain, Humain vs Machine, Machine vs Machine en respectant la structures des données (cf p. 4) et les règles du jeu.

Le Chicago Stock Exchange est un jeu de société qui utilise un plateau. Ce dernier se base essentiellement sur des principes de finance et de bourse: moins on vend une marchandise, plus elle se raréfie et devient chère. Les joueurs s'alternent en déplaçant successivement un trader. Ils récupèrent deux céréales (respectivement à gauche et à droite de la pile où est positionné le trader) à chaque tour, gardent une céréale et vendent l'autre sur le marché boursier. La vente décrémente la valeur en bourse de la céréale d'une unité. Le but du jeu est d'avoir le score le plus élevé, la stratégie étant de s'enrichir au maximum tout en appauvrissant l'adversaire.

Structures de données

- Le plateau : une liste composée de la liste des piles de céréales, de la position du Trader, de la bourse, de la réserve du Joueur 1 et de la réserve du Joueur 2.
  [PilesMarchandises, PositionTrader, Bourse, RéserveJoueur1, RéserveJoueur2]

- Les piles du plateau : une liste composée de sous listes. Initialement, il y a 9 piles.
  P = [P1, P2, P3, P4, P5, P6, P7, P8, P9]

- Une pile : une liste composée de 4 éléments.
  [A, B, C, D]

- Le Trader : il est représenté par un entier. Initialement, c'est un nombre compris entre 1 et 9.
  PositionTrader

- La Bourse : une liste composée de 6 sous-listes.

Ces sous-listes sont de la forme [Céréale, Valeur]. Initialement, la bourse est présentée ainsi :
  [[ble,7],[riz,6],[cacao,6],[cafe,6],[sucre,6],[mais,6]]

- Les réserves des joueurs : ce sont des listes initialement vides. Ensuite, elles se remplissent avec les céréales gardées par chaque joueur.
  [RéserveJoueur1]
  [RéserveJoueur2]

Principaux prédicats

Prédicats d'affichage et d'initialisation

Générer le plateau :

generer_piles(P1, P2, P3, P4, P5, P6, P7, P8, P9, P) :

D'abord nous commençons par générer les piles. Pour cela nous utilisons le prédicat generer_piles associé à jetons (composé de 6 jetons de chaque marchandise) et une_pile, ce dernier consiste à choisir aléatoirement un jeton (prédicat choose), l'insérer dans une pile puis supprimer cet élément (à l'aide du prédicat del) de la liste de jetons, afin d'obtenir exactement 6 répétitions de chaque marchandise dans les 9 piles. Chaque pile est constituée de 4 éléments lors d'une itération d' « une_pile » dans le prédicat generer_piles, le résultat est donc une liste de 9 sous-listes.

Pour l'affichage des piles, nous choisissons d'afficher les têtes de chaque sous liste (en suivant les règles du jeu dans lesquelles les joueurs ne voient que la pile du dessus) accompagnées de leurs indices (positions dans la liste des 9 piles). Ceci nous permettra d'afficher un trader, en unifiant sa position avec l'indice de la pile correspondante. Le prédicat positionInitiale nous permet de générer aléatoirement un trader, et affiche_piles_ini_trad permet l'affichage des piles souhaitées.

Afficher bourse :

affiche_bourse(B)

Pour le plateau de départ, on initialise la bourse dans le prédicat bourse avec les valeurs citées dans les règles du jeu.

En ce qui concerne son affichage, nous utilisons affiche_bourse, qui utilise dans son déroulement afficheValeur. Il permet une présentation cohérente de la bourse dans notre plateau à l'aide de ecrire.

Afficher les réserves :

Nous utilisons le prédicat affiche_J1Reserve et affiche_J2Reserve pour afficher les réserves des deux joueurs. Celles du plateau de départ sont initialement vides .

Afficher le plateau :

Enfin pour l'affichage du plateau, nous utilisons deux prédicats :

-plateau_depart(M, Pos, B, R1, R2) : il initialise le plateau avec les piles générées au tout début d'une partie avec le prédicat affiche_piles_ini_trad

-plateauEncours(M, P, B, J1R, J2R) : il affiche l'état courant d'un plateau au fil du jeu, c'est à dire qu'à l'aide du prédicat affiche_piles, il ne réinitialise pas les piles depuis leur génération, mais récupère celles du plateau en cours et prend en considération la nouvelle position du trader. Il se comporte de la même manière avec la bourse, nous n'unifions plus B avec bourse(B) car on a besoin des valeurs de la bourse courante après sa modification.

Prédicats pour le jeu humain

coup_possible(+Plateau, ?Coup) :

Ce prédicat a pour but de déterminer si un coup est possible à partir d'un plateau donné en paramètre. Il faut déterminer le déplacement du trader, la céréale gardée et la céréale vendue par le joueur effectuant ce coup.

Tout d'abord nous testons avec une boucle repeat si le déplacement rentré par l'utilisateur est valide(1, 2 ou 3). Si le déplacement est valide, nous l'ajoutons à la position actuelle.
Ensuite, nous effectuons un modulo sur cette nouvelle position afin de gérer la dimension circulaire des piles du plateau.

Ce prédicat modulo a été redéfini afin d'obtenir des résultats compris entre 1 et le nombre de piles.
(Exemple : si on est à la position 7, avec un nombre de piles égal a 9, qu'on se déplace de 2, la nouvelle position est 9. Le modulo usuel nous renverra la valeur 0, ce qui n'est pas cohérent avec notre jeu, nous aurons besoin dans ce cas d'un résultat égal à 9).

Par ailleurs, nous calculons les positions précédente et suivante de la nouvelle position du trader (prédicats positionPrec et positionSuiv). A partir de ces deux indices, nous cherchons les têtes des deux sous listes correspondantes à ces indices.
Ces deux céréales sont ensuite proposées à l'utilisateur afin qu'il choisisse celle qu'il souhaite garder, et celle qu'il souhaite vendre à la Bourse. Son choix est utilisé dans le prédicat cerealegardee.

Prédicats pour l'Intelligence Artificielle

coups_possibles(+Plateau, +Joueur, -ListeCoupsPossibles) :

Ce prédicat a pour objectif de lister tous les coups possibles à partir d'un plateau de jeu donné.

A chaque tour de jeu, 6 possibilités s'offrent à un joueur : 3 déplacements différents * 2 choix possibles pour les céréales gardée et vendue.

Afin de déterminer ces coups_possibles, on a utilisé le prédicat coup_possible_ordi basé sur le prédicat coup_possible sauf qu'aucune interaction n'est possible avec l'utilisateur (c'est une IA). Nous déterminons alors la céréale gardée et la céréale vendue. Ceci est réalisé pour les 3 déplacements possibles, en inversant à chaque fois les céréales gardées et vendues.

Nous obtenons à la fin les 6 coups que nous insérons dans une liste qui est renvoyées en sortie du prédicat.

meilleur_coup(+Plateau, -Coup) :

Ce prédicat a pour objectif de déterminer le meilleur_coup a jouer pour l'IA à partir d'un plateau donné.

Nous avons utilisé la stratégie d'**enrichissement** pour le joueur effectuant ce coup.
Tout d'abord, nous appelons le prédicat coups_possibles défini précédemment. Nous obtenons les 6 coups possibles. Ensuite, nous simulons chacun de ces coups pour déterminer le score à l'issu de

ces coups (prédicat simuler_coup_ordi). Il ressemble fortement à jouer_coup sauf qu'il n'affiche rien, ne modifie pas le plateau mais il renvoie le score du coup simulé.
Avec ces 6 scores, nous déterminons le score maximum (prédicat maximum_liste). Avec ce score, nous revenons au coup associé qui est renvoyé en sortie du prédicat. Nous avons alors le meilleur coup avec une stratégie d'enrichissement.

Prédicats communs aux jeux Humain et IA

jouer_coup(+PlateauInitial, ?Coup, ?NouveauPlateau) :

Ce prédicat permet de déterminer l'état du jeu (NouveauPlateau) après application d'un coup.

Comme pour coup possible, nous déterminons la nouvelle position du trader en ajoutant le déplacement à la position initiale.
Il faut maintenant ajouter la céréale gardée à la réserve du joueur concerné. Pour réaliser cette opération, on utilise le prédicat add_reserve qui utilise lui-même le prédicat add (ajout en début de liste d'un élément). La réserve du joueur non cerné reste inchangée.

Ensuite, la bourse doit être modifiée en décrémentant la valeur de la céréale vendue. Nous utilisons le prédicat bourse_sortie. Il recherche la paire [Vend,Valeur] dans la bourse (prédicat element). La valeur est décrémentée de 1, puis on substitue cette nouvelle paire [Vend, Valeur] dans la bourse que l'on renvoie en résultat.

Enfin, la dernière chose restante à pourvoir est de modifier le plateau de jeu. Il faut retirer les 2 jetons utilisés lors de ce coup. Pour cela, le prédicat delete_first_sous_liste va supprimer le premier élément des sous listes d'indice Prec et Suiv.
Pour terminer, on supprime les éventuelles liste vides (delete_element) qui deviennent inutiles pour la suite du jeu.

score_joueur(+Joueur, +Bourse, +J1R, +J2R, ?Score) :

Ce prédicat a pour objectif de déterminer le score d'un joueur à partir de la Bourse à un état donné.

On unifie tout d'abord le Joueur avec j1 ou j2 pour savoir quelle réserve sera utilisée pour calculer le score. Ensuite, le prédicat score_reserve est appelé. Il utilise lui-même score_Elt qui permet d'avoir la valeur d'une céréale. Les valeurs de chaque céréale sont ajoutées et on obtient le score du joueur.

qui(?Joueur) :

Ce prédicat permet de déterminer quel joueur va commencer à jouer. Il est appelé au début des boucles de jeu. « Joueur » est unifié avec la valeur « j1 » ou « j2 ». La question est répétée si la valeur saisie est différente.

alterner(+JoueurActuel, ?JoueurSuivant) :

Ce prédicat prend en entrée le joueur actuel (« j1 » ou « j2 ») et renvoie l'autre joueur dans JoueurSuivant.

gagnant(+ScoreJoueur1, +ScoreJoueur2) :

Le prédicat gagnant permet de déterminer quel joueur a gagné. Il prend en entrée les scores des 2 joueurs et renvoie un commentaire avec le joueur gagnant, ou les deux s'ils sont à égalité.

Boucle Menu

Cette boucle répète le menu. Nous nous sommes inspirés du cours. Ce menu est un prédicat qui demande à l'utilisateur quelle type de partie il souhaite effectuer parmi les 3 possibles. En fonction de la réponse donnée par l'utilisateur, un appel est effectué. Il lance le prédicat correspondant au type de jeu : jVSj, iaVSia et jVSia. Nous avons également donné la possibilité de quitter le jeu.
Le cut à la fin de chaque appel éviter de rappeler « Vous avez mal choisi » lorsqu'on a saisi un choix valide.

Boucle Humain/Humain :

Cette boucle permet de dérouler une partie entre deux joueurs humains. On y utilise donc le prédicat coup_possible et jouer_coup. jVSj initialise le plateau de départ, demande au début quel joueur commence puis lance la partie entre les joueurs. On alterne les tours des joueurs grâce au prédicat alterner : il prend j2 si le joueur courant est j1, et vice versa. Le prédicat boucle_JvsJ est implémentée de telle sorte qu'il sorte de la partie si le nombre de piles restantes dans le plateau courant est inférieur ou égal à deux, pour calculer les scores finaux et déterminer le gagnant. Sinon, la boucle lance le premier tour du joueur choisi avec son coup et son exécution, détermine le joueur suivant, affiche le nouveau plateau et relance la boucle sur le deuxième tour su joueur suivant, ainsi jusqu'à là condition d'arrêt

Boucle IA/IA :

Elle est basée sur le même principe que la boucle Humain/Humain.
Un premier prédicat iaVSia affiche le plateau_depart et demande quel est le joueur qui commence. Il lance ensuite la boucle_IAvsIA. Ce prédicat cherche le meilleur_coup à effectuer et l'exécute grâce à jouer_coup. Ensuite, nous changeons de joueur avec alterner, affichons le nouveau plateau de jeu (prédicat plateauEncours) et relançons la boucle_IAvsIA.
Si la condition nombre de piles <= 2 est vraie, le jeu doit se terminer.
On calcule alors le score des deux joueurs à partir de la bourse à la fin du jeu (score_reserve). Enfin, nous déterminons quel joueur a gagné, c'est-à-dire celui qui a le score le plus élevé (prédicat gagnant).

Boucle Humain/IA :

Cette boucle est « une fusion » des deux précédentes boucles.
L'Humain débute la partie (prédicats coup_possible et jouer_coup) et l'IA joue juste après grâce au prédicat suite_boucle_IA. Ce dernier rappelle alors boucle_JvsIA.

On teste la fin du jeu après chaque joueur (suite_boucle_IA). Si le nombre de piles est inférieur ou égal à 2, on exécute la même chose que dans les 2 autres boucles, c'est-à-dire calculer les scores des joueurs et déterminer le gagnant.

Difficultés rencontrées

- L'affichage du Trader nous a posé quelques soucis au début du projet. Nous avions tout d'abord affiché le trader sous les piles avec un « X » pour le représenter. Or, un des problèmes était de gérer le nombre d'espaces entre chaque pile. En effet, chaque pile a une taille différente (on affiche la céréale sur la pile). De plus, nous avions d'autres soucis avec la suppression des piles au cours du jeu. Le Trader avait une positon supérieure au nombre de piles du plateau. De ce fait, il ne s'affichait plus : nous avons donc implémenté le prédicat repositionne_trader. Nous avons alors opté pour une autre solution. La position du Trader s'unifie avec le numéro de la pile correspondant.

```
-------Position du Trader : 6 -------
1  ble
2  cafe
3  sucre
4  mais
5  sucre
6  cacao           <-- Trader
7  mais
8  cacao
9  riz
```

- La décrémentation du nombre de piles. Il fallait prendre en compte le fait que les piles se vident au fur et à mesure du jeu. De ce fait, certaines piles deviennent vides. Nous avons donc implémenté un prédicat qui élimine ces piles du plateau.

- La détection des erreurs. En effet, il nous est arrivé de chercher pendant un moment avant de découvrir d'où venaient certaines erreurs. Il a fallu tester les prédicats séparément et afficher des commentaires pour cibler exactement le problème. La fonctionnalité trace n'est pas très pratique dans le cas des boucles où plusieurs prédicats sont appelés.

- Certains prédicats ont du être modifiés plusieurs fois pendant ce projet. Certains ne prenaient pas en compte des cas spécifiques et il fallait alors revenir sur ces prédicats pour pouvoir avancer.

Améliorations possibles

- Algorithme Minimax à implémenter afin d'avoir une Intelligence Artificielle mieux gérée. En effet, nous ne regardons que les 6 coups possibles afin de déterminer le meilleur coup de l'IA. Avec cet algorithme, on aurait pu déterminer beaucoup plus de coups possibles en se basant sur une profondeur de recherche. A partir de celle-ci, en modifiant la valeur de la profondeur, il aurait été possible de fixer des niveaux de difficultés pour l'IA.

- Dans une moindre mesure, nous aurions pu gérer la différence de scores entre les deux joueurs au lieu de gérer seulement le meilleur score du joueur à un instant T. Il aurait été plus judicieux et intéressant pour les mécaniques du jeu de calculer le score de l'adversaire pour chaque coup, et calculer la différence des scores. Ainsi, on aurait choisi la différence de scores la plus élevée, et on aurait ou définir un prédicat meilleur_coup plus efficace.

- Nous avons essayé de gérer les erreurs de saisie par l'utilisateur mais ce n'est pas parfait. Le choix du joueur a été parfaitement réalisé mais pour le choix du déplacement, seules les valeurs numériques sont gérées. Il aurait fallu pouvoir vérifier tous les types de caractères.

- Interface graphique. Nous avons tenté de rendre le jeu le plus attrayant possible tout en assurant les fonctionnalités du jeu. L'interface est améliorable, par exemple en affichant éventuellement le nombre de jetons restant par pile.

- Créer plus de prédicats pour éviter la redondance dans le code, et donc le condenser. Le langage prolog se basant sur des prédicats, il aurait été plus judicieux de créer plus de sous prédicats, et ainsi diminuer la taille des prédicats importants. Mais ceci présente une contrepartie car il faut se déplacer de prédicat en prédicat pour détecter les erreurs éventuelles.

Conclusion

Ce projet a été très intéressant sur différents points. Tout d'abord, il nous a permis de mettre en application nos connaissances en programmation logique vues dans le cadre de l'UV IA02.

Ensuite, ce projet a été l'occasion d'utiliser la langage Prolog. Ce dernier est très puissant et permet de réaliser des actions avec un nombre d'instructions très réduit grâce à la récursivité.

Par ailleurs, réaliser un jeu entre deux joueurs est un sujet très ludique. Nous avions hâte d'arriver au bout de ce projet pour pouvoir tester ce jeu et voir le fruit de notre travail.

Enfin, la réalisation d'une Intelligence Artificielle est quelque chose de tout nouveau pour nous. Nous avons pu découvrir les différentes méthodes pour l'implémenter (algorithme Minimax, élagage α-β, enrichissement, appauvrissement…).