IA02 – « Chicago Stock Exchange »

Théophile GINDRE

Théo JUDAS

- P15 -

Table des matières

I/ Présentation du jeu

Le projet que l'on a développé porte sur le jeu « Chicago Stock Exchange ». L'objectif est de coder ce jeu dans le langage déclaratif PROLOG afin d'en exploiter les capacité. Il doit mettre à disposition 3 modes de jeu : « Humain vs Humain », « Humain vs Ordinateur » et « Ordinateur vs Ordinateur ».

Voici un visuel du produit fini :

```
| ?- lancer_jeu.

      **************************************************
      ************* Chicago Stock Exchange *************
      **************************************************

******Menu*****

 1 - Partie Humain VS Humain
 2 - Partie Humain VS Machine
 3 - Partie Machine VS Machine

****************

Votre choix : 1.

Qui commence ? (j1 ou j2) : j1.

****Plateau****

 Marchandises :
  Trader ->  1 : ble,cafe,riz,sucre
             2 : cafe,cacao,cacao,mais
             3 : mais,sucre,ble,mais
             4 : sucre,cacao,riz,ble
             5 : ble,cacao,riz,riz
             6 : mais,cafe,ble,ble
             7 : riz,cafe,sucre,sucre
             8 : cafe,sucre,riz,mais
             9 : cacao,cafe,cacao,mais

 Bourse : [ble,7],[riz,6],[cacao,6],[cafe,6],[sucre,6],[mais,6]
 Reserve joueur 1 :
 Reserve joueur 2 :

****************

j1 - Entrez un deplacement entre 1 et 3 :
```

II/ Prédicats utilisés

Dans un premier temps, seront exposés les principaux prédicats du jeu, leur fonctionnement et comment ils interagissent ensemble dans le cadre du projet.

A- Initialisation et gestion du jeu

lancer_jeu

Ce prédicat est utilisé pour lancer une partie du jeu « Chicago Stock Exchange ». Il commence par afficher le menu, qui propose trois modes de jeu : « Humain vs Humain », « Humain vs Ordinateur » et « Ordinateur vs Ordinateur ». Après avoir récupéré dans une variable le choix de l'utilisateur, il appelle le prédicat *mode* qui prend en paramètre ce choix.

mode

Quel que soit le mode de jeu choisi par l'utilisateur, mode commence par initialiser aléatoirement le plateau de départ (voir description ci-dessous). Ensuite, pour débuter la partie, on demande à l'utilisateur de choisir qui du joueur 1 et du joueur 2 aura la primeur de commencer, grâce à *read_joueur*. Les propositions varient suivant le mode de jeu pour afficher un choix explicite comme « Ordinateur » au lieu de « Joueur » suivant le mode.

On lance ensuite le prédicat *jouer*, qui prendra en paramètre le plateau de départ, le mode de jeu, le Joueur qui commence et le plateau d'arrivée, qui sera calculé suivant le déroulement de la partie (voir description ci-dessous).

Une fois le plateau final unifié, le prédicat calcule le score des deux joueurs et affiche le gagnant suivant ces scores, avec un message personnalisé en fonction du mode de jeu.

plateau_depart

plateau_depart permet d'initialiser le plateau de départ en mélangeant aléatoirement les ressources dans 9 piles de 4 ressources. Pour cela, il unifie la variable en paramètre avec une liste de type : *[Piles, Bourse, PositionTrader, ReserveJ1, ReserveJ2]*, d'après la représentation choisie.

- *Bourse* est initialisée avec la liste : [[ble,7],[riz,6],[cacao,6],[cafe,6],[sucre,6],[mais,6]].
- *PositionTrader* est initialisé à 0.
- *ReserveJ1* et *ReserveJ2* sont initialisés avec la liste vide.
- *Piles* est calculée par l'appel successif à *shuffle* qui mélange aléatoirement les éléments d'une liste et à *cut4* qui divise une liste en sous listes de 4 éléments. Ainsi, Piles contiendra 9 piles de 4 éléments, distribués aléatoirement.

affiche_plateau

Le plateau est le seul paramètre. On délimite bien l'affichage du plateau pour que l'utilisateur se repère facilement. On affiche une à une les piles de marchandises, numérotées entre 1 et le nombre de piles non-vides sur le plateau. De plus, à gauche de la pile où se trouve le Trader, on ajoute une indication fléchée montrant que le Trader est actuellement sur cette pile. Est ensuite affichée la bourse, avec les valeurs actuelles des ressources, suivie par la liste des ressources possédées par chacun des deux joueurs.

jouer

Ce prédicat récursif correspond au déroulement de la partie. Il prend en paramètre le plateau de départ et unifie le dernier paramètre avec le plateau final, lorsqu'il ne reste plus que 2 piles de ressources ou moins (condition d'arrêt de la partie).
Le prédicat *jouer* est légèrement différent en fonction du mode de jeu choisi :

- En mode « Humain vs Humain », il demande au joueur à qui c'est au tour de jouer (variable Joueur, passée en paramètre) de choisir un déplacement entre 1 et 3. Puis il fait appel à *coup_possible* qui unifie R1Tmp et R2Tmp avec les deux ressources qu'il lui est possible de choisir à l'issue de ce déplacement (Cf. description ci-dessous). Il continue en proposant au joueur de faire un choix entre ces deux ressources pour savoir laquelle il souhaite garder, puis il fait appel à *jouer_coup*, qui va jouer le coup correspondant aux deux choix du joueur (déplacement et ressource à garder). Enfin, il va appeler récursivement *jouer* en indiquant que c'est au tour du joueur adverse.

- En mode « Ordinateur vs Ordinateur », le déroulement est identique, sauf qu'à la place d'attendre une entrée au clavier de la part des joueurs, c'est la fonction *meilleur_coup* qui détermine le coup que l'ordinateur jouera (la fonction choisira le coup qui semble le plus à son avantage, Cf. II/).

- En mode « Humain vs Ordinateur », les deux types de déroulement se feront alternativement.

B- Jouer un coup

prises_possibles

Le prédicat unifie le dernier paramètre avec la liste des deux couples de ressources qu'il est possible de choisir. On prend également en compte la position du Trader après son déplacement, et l'état actuel des piles. On calcule les positions des piles à gauche et à droite du Trader, suivant le nombre de piles encore en jeu. Les deux couples de prises possibles, « ressource gardée/ressource jetée », sont les premiers éléments de chacune des deux listes.

coup_possible

Ce prédicat vérifie que le coup indiqué en paramètre est valide. Pour cela, il appelle *prises_possibles* qui renvoie la liste L des deux couples de ressources autour de la position actuelle à laquelle on a ajouté le déplacement. Il vérifie ensuite que la liste « ressource gardée/ressource jetée » appartient à L.

En réalité, *coup_possible* est utilisé en génération dans notre projet. Nous l'utilisons pour récupérer les deux ressources qui entourent la position de destination du trader, et non pas pour vérifier que le coup est valide.

positionner_trader

Avec posititionner_trader, on fait d'une pierre deux coups. En effet, en distinguant quatre situations, on va enlever les piles vides après avoir joué un coup et repositionner le Trader si des piles. Pour cela, on prend la liste des piles, la position du Trader d'une part, la liste des piles sans les listes vides et la nouvelle position du trader d'autre part.

- Si on ne se trouve ni à droite ni à gauche du Trader, on ne modifie ni la pile ni la position du trader.
- Si on est sur la pile gauche ou droite, qui a un indice inférieur à la position du trader, on retire la liste vide et on décrémente la position du trader.
- Si on est sur une pile vide avec un indice supérieur à la position du Trader, on retire simplement la liste vide.
- Le prédicat étant récursif, on s'arrête quand il n'y a plus aucune pile à tester.

score

Nous calculons le score de manière récursive. Nous allons prendre une à une les ressources de la réserve d'un joueur et lui attribuer un chiffre correspondant à sa valeur d'après la bourse finale. Nous nous arrêtons une fois qu'il ne reste qu'une liste vde dans la réserve et nous renvoyons 0 comme score, qui initialise le score à 0. En remontant les appels récursifs, on ajoute les valeurs de chaque ressource pour obtenir le score total.

jouer_coup

jouer_coup sert à jouer le coup qui est renseigné en paramètre. Etant donné que *coup_possible* est appelé avant, on considère que le coup est valide. Le prédicat commence donc par calculer le numéro de la pile vers laquelle on doit se déplacer, puis appelle les différents prédicats chargés de modifier le plateau de jeu correctement.

- *modif_piles* retire les deux ressources concernées (gardée et jetée) des piles du plateau.
- *positionner_trader* retire les piles vides en repositionnant éventuellement le trader (Cf. ci-dessous).
- *decrementer_bourse* met à jour les valeurs des ressources dans la bourse.
- *maj_reserves* ajoute la ressource gardée à la réserve du joueur concerné.

III/ Description de l'IA

Voici les quatre prédicats principaux permettant à l'ordinateur de choisir le meilleur coup à jouer.

coups_possibles

Pour ce prédicat, le but recherché est de retourner l'ensemble des 6 coups possibles à effectuer pour un joueur, en fonction du déplacement (entre 1 et 3) et du choix de la ressource à garder (2 choix pour chaque déplacement). Nous procédons en renseignant les trois positions du joueur possible après son déplacement. Puis, on associe à chacune des positions les deux couples de prises possibles en nous servant du prédicat préalablement défini, *prises_possibles*. Chaque couple de prises possibles est alors associé à un coup, composé en plus du Joueur et du déplacement correspondant. On obtient alors facilement la liste des six coups possibles.

variation_score

variation_score est identique au prédicat score défini précédemment, à ceci près qu'on ne s'intéresse qu'aux valeurs du couple de ressources d'un coup. Ainsi, on ne calcule que les valeurs qui changent si le coup est réalisé, ce qui permet d'apprécier la variation du score.

attribuer_score

L'objectif de ce prédicat est d'attribuer un score (une note) à chacun des coups contenus dans le deuxième paramètre du prédicat. Plus le score attribué est élevé, plus le coup sera bénéfique pour l'ordinateur qui le jouera. Pour calculer ce score, on simule le coup en appelant *jouer_coup* et en récupérant le plateau de jeu ainsi modifié. On appelle ensuite *variation_score* deux fois pour calculer le score des deux joueurs, puis *affecter_score* calcule la différence de score entre les deux joueurs. C'est cette différence de score qui constitue la note attribuée au coup.
Le prédicat finit en s'appelant récursivement sur la suite des coups à analyser.

meilleur_coup

Ce prédicat retourne le meilleur coup à jouer pour l'ordinateur sur le plateau courant. Il fait appel à *coups_possibles* pour récupérer la liste des 6 coups possibles, puis appelle *attribuer_score* pour leur attribuer à chacun une note. C'est la note la plus élevée qui sera retenue, c'est pourquoi on utilise le prédicat *max_score* qui retourne le score maximum et le prédicat *index_coup* qui retourne son index dans la liste. On utilise enfin *nth0* pour sélectionner le coup correspondant et l'unifier avec le meilleur coup à jouer.

Somme toute simple, l'intelligence artificielle développée choisit le coup adéquat certes, mais ne prévoit pas plusieurs coups à l'avance, en tenant compte par exemple des ressources qui ne sont pas encore les premières de leur pile, ou encore du prochain déplacement possible de l'adversaire.

IV/ Difficultés rencontrées et améliorations possibles

Tout d'abord, appréhender le fonctionnement d'un langage de programmation déclarative fut compliqué. Prolog a une manière de fonctionner singulière, qui ne ressemble en rien à ce qu'on a pu apprendre auparavant. Il nous a fallu un peu de temps et quelques déclics, puis le développement du jeu est allé très rapidement. La seule autre difficulté rencontrée a été le fait que les variables sont statiques, on ne peut les unifier qu'une seule fois. Nous avons appris après avoir fini le projet qu'il était possible de les rendre dynamique, cependant nous avons pallié le problème sans avoir recours à l'unification dynamique. Le travail en binôme fut très équilibré, nous nous sommes autant investi l'un que l'autre, et avons participé à chaque étape du développement du jeu.

En regardant notre code et les alternatives proposées par l'unification dynamique, il nous est apparu que dans le cas précis du développement de ce jeu, le dynamisme n'apportait pas une vraie valeur ajoutée. Cependant, dans le cadre d'un jeu plus complexe, nous pensons qu'avoir recours à l'unification dynamique des variables serait pertinent. De plus, nous pensons très fortement que l'IA de l'ordinateur peut être largement améliorée, en instaurant des niveaux de difficulté en plus par exemple.