**Présentation du jeu : et analyse ? Mettre les structures de données. Comment on a traité les données du problème**

**Une présentation des prédicats principaux composant le jeu.**

Prédicats d’affichage :

**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 précédentes et suivantes 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 predicat 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 gardées et vendues.

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.

**Predicats 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 « j 1 » 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.

**La description de l’IA et des choix faits au niveau de l’implémentation et de l’évaluation des coups.**

**Les difficultés rencontrées et améliorations possibles**

Algo 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.

La décrémentation du nombre de piles. Il faut prendre en compte le fait que les piles se vident au fur et à mesure du jeu. De ce fait, certaines piles deviennent : il faut donc les éliminer du plateau pour que le trader ne pointe plus sur des piles vides.

Interface graphique.

Prédicats d’affichage et d’initialisation :

Générer le plateau:

**generer\_piles :**

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 joueur ne voit 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 souhaité

positionTrader ???

**afficher\_bourse :**

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 également 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.

**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 predicat alterner : il prend j2 si le joueur courant est j1, et vice versa. Le predicat boucle\_JvsJ est implémentée de telle sorte qu’il sorte de la partie si le nombre de piles restantes dans le plateau courant est inferieur 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