VUATRIN Artus
JEANNOT Antoine

Printemps 2015
Université de Technologie de Compiègne

IA02

Chicago Stock Exchange

Table des Matières

Présentation des principaux prédicats

Affichage du plateau

La fonction d'affichage du plateau afficherPlateau() appelle les différentes fonctions d'affichage pour les différents éléments du plateau.

Pour afficher la bourse (prédicat afficherBourse()), on écrit le nom d'une réserve puis on écrit sa valeur, et on fait appelle récursivement à la fonction pour afficher le reste de la liste bourse.

Pour les piles, nous avons choisi d'afficher le premier élément de chaque pile horizontalement avec le prédicat afficherPiles() chaque représentation d'une pile occupe 7 caractères, ainsi la taille occupée par l'affichage des piles est constant. L'écriture de la position est rendue possible grâce à cela. On ajoute (7* (position-1) +3) espaces devant le caractère représentant le pion du joueur. Le pion est ainsi placé en dessous de la pile sur laquelle il se trouve grâce au prédicat afficherPosition()

L'affichage des réserves est simplement un appel récursif au prédicat afficherReserve() sur la liste représentant les réserves pour écrire le nom de chaque ressource.

Déroulement du jeu

Au démarrage d'une partie, on demande au joueur le mode de jeu (le type des joueurs : IA ou humain) puis la difficulté de la partie (qui représente la profondeur de recherche de l'IA) avant de lancer la boucle correspondant au type de jeu.

La boucle principale du jeu est gérée par le prédicat et se compose de :

- Affichage du Plateau
- Tour du joueur 1
- Affichage du plateau résultant des modifications apportées par le coup du J1
- Tour du joueur 2

Un tour d'humain contrôlé par le prédicat playerTurn() se compose de :

- Vérification de fin de partie
- Exposition des différents coups possibles
- Modification du plateau selon la demande du joueur

Un tour d'IA est contrôlé par le prédicat computerTurn() se déroule ainsi :

- Vérification de fin de partie
- Recherche du meilleur coup possible, selon la difficulté définie par l'utilisateur
- Modification du plateau selon le coup trouvé précédemment

Pour vérifier si une partie est finie (prédicat fin()), on compte le nombre de piles restantes: s'il en reste moins de deux, la partie est terminée et on appelle le prédicat score() qui calcule le score d'un joueur en retrouvant chaque ressource de la réserve d'un joueur dans la bourse afin de calculer la valeur de l'ensemble de la réserve. Puis le prédicat félicitation est appelé pour afficher les résultats finaux.

Préparation du coup

Le prédicat prepare_coup() est le contrôleur du jeu qui permet au joueur d'effectuer un tour de jeu. Il demande par l'intermédiaire de deplacerPion() le nombre de piles à avancer, et boucle en cas de réponse impossible ou erronée. Puis demande au joueur la ressource qu'il souhaite conserver et celle qu'il désire jeter grâce à choixRessource(), qui recherche le sommet des piles adjacentes en utilisant les prédicats ressource() et nth1(). Le prédicat ressource() sélectionne les piles adjacentes au pion et le prédicat nth1() est utilisé pour sélectionner une ressource à afficher. Pour finir, le prédicat ecrireRessource() renvoie une liste de deux éléments où le premier élément est la ressource à conserver et le second élément est la ressource à rejeter.

Application du coup

Après avoir trouvé tous les paramètres permettant de former un coup, il faut effectuer les modifications sur le plateau. C'est ce que jouer_coup() s'occupe de faire. On doit d'abord modifier les piles. Pour cela le prédicat updatePile() retire les ressources utilisées des piles adjacentes à la position du joueur, puis retire les piles vides grâce au prédicat clean(). Ensuite, les prédicats correctionPosition () et correctionPositionBeforePlayer() se chargent de modifier la position du joueur en fonction des piles supprimées respectivement après et avant sa nouvelle position.

Ensuite jouer_coup() met à jour la bourse en retirant un point de valeur à la ressource défaussée avec updateBourse() et le prédicat ajouterRessource() met à jour les listes de ressources du joueur en ajoutant la ressource à conserver à sa réserve.

Conception de l'Intelligence Artificielle

L'intelligence artificielle du jeu a été conçue selon l'algorithme du « MinMax ».

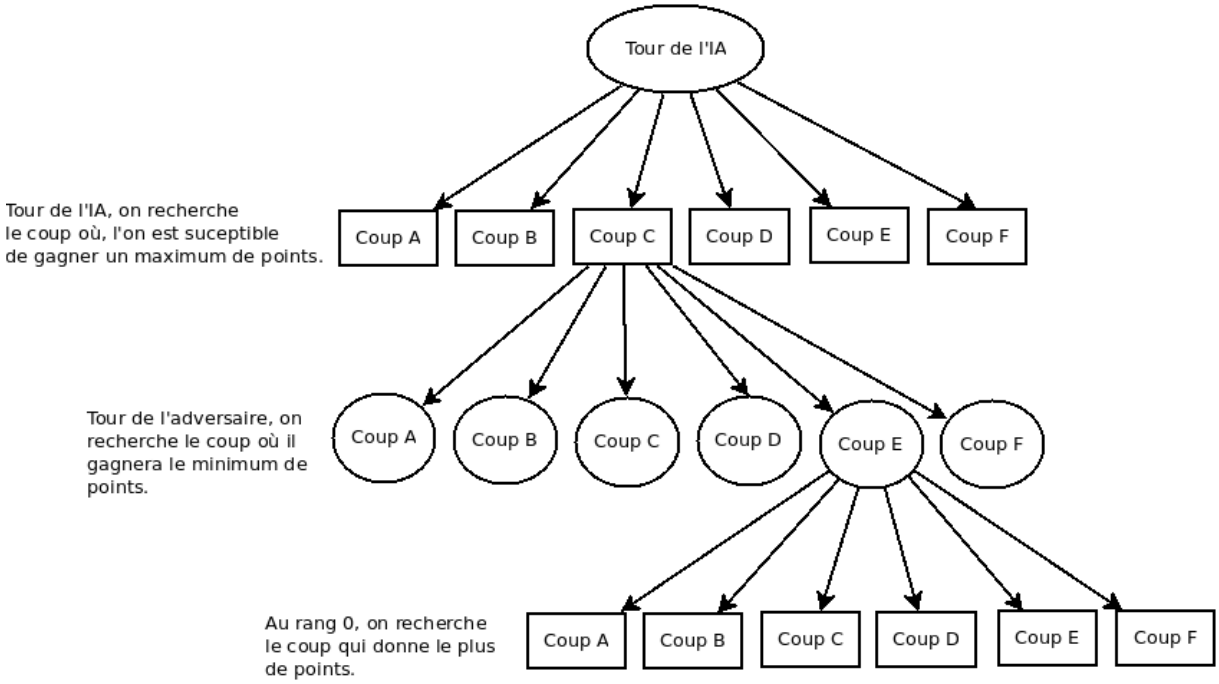
Le nombre de coups possibles à rechercher évoluant de manière exponentielle, une variable difficulté a été déclarée et est définie par l'utilisateur en début de partie. Cette variable correspond en réalité au niveau de profondeur, de « vision, de l'Intelligence Artificielle. Si la difficulté est réglée au maximum, soit sur trois, cette dernière recherchera le meilleur coup possible à jouer au tour n pour obtenir le meilleur score à l'issu du tour n+2.

Coups_possibles() est le prédicat qui recherche tous les coups possibles et les renvoie dans une liste. Pour cela, on commence par rechercher les trois déplacements possibles, avec le prédicat liste_deplacementsIA(). Pour chacun de ces déplacements, on concatène les deux possibilités de choix des ressources grâce au prédicat choixRessourcesIA(). On obtient ainsi six coups possibles.

MeilleurCoupSup() se charge de trouver le meilleur coup parmi la liste trouvée grâce au prédicat précédent. Pour cela, on va simuler chacun des six coups trouvés et maximiser le résultat en considérant que l'adversaire va aussi essayer de trouver le meilleur coup possible pour maximiser son score. Ainsi on va essayer de minimiser le score de l'adversaire tout en maximisant le score de l'IA. La difficulté représente la profondeur de recherche, elle est diminuée à chaque appel récursif de MeilleurCoupSup(). Si cette valeur tombe à 0, c'est que on ne veut plus chercher plus loin, on renvoie donc le score de l'IA qui est obtenu à cet état. Si c'est l'IA qui appelle la fonction MeilleurCoupSup() alors c'est que c'est son tour de jouer, on recherche donc le meilleur score parmi ceux proposés en utilisant le prédicat maxList(). Il se charge de renvoyer le coup qui fait le score maximal. Sinon, c'est au tour de l'adversaire et on minimise son score à l'aide du prédicat minList(). Dans le cas où le coup joué déclenche le prédicat fin(), on cherche alors le meilleur coup possible avec le prédicat meilleur_coup().

Le prédicat meilleur_coup() appelle recherche_meilleur_coup() sur tous les coups possibles. Ce dernier va simplement maximiser le score du joueur en calculant le score de chaque coup possible.

Sur la figure suivante, on peut voir l'exemple d'une recherche de coup pour une difficulté égale à 2.



Difficultés rencontrées

La difficulté majeur rencontrée a été la compréhension des concepts et algorithmes d'Intelligence Artificielle vus en cours et ré-expliqués par M. Carpentier, ainsi que leur intégration au sein de l'architecture de notre programme et de nos prédicats.

Egalement, l'apprentissage de la syntaxe du Prolog et de sa représentation sous forme de liste a été quelque peu difficile à comprendre et à « bien » utiliser en début de parcours.

Optimisations

A l'issu de ce projet, nous nous sommes rendus compte que nous pouvions améliorer ce programme au delà des exigences imposées au sein du cahier des charges.

Voici quelques possibilités d'optimisation possibles qui nécessiteraient relativement peu de modifications de l'architecture actuelle, les nouveaux « modules » complémentaires consisteraient en des « sur-couches » ajoutées à notre moteur:

- Lancer des parties à N joueurs, N >= 2 (dans le domaine du raisonnable et de la pertinence du jeu évidemment, un multi-joueur à plus de 4 personnes commence à perdre de son intérêt mais est possible).

- Concevoir différents types de stratégie, les nommer sous forme de héros, et permettre à l'utilisateur de les choisir en début de partie (exemple: jouer contre Bowser, Tyrion, Bilbo..) afin de varier le caractère des parties.

- Mettre en place un mode histoire pour rendre le mode « un joueur » plus intéressant.

- Installer place un système de sauvegarde des parties et des scores avec une exportation dans un fichier texte afin de pouvoir arrêter une partie et la reprendre plus tard. On pourrait éventuellement rejouer une partie déjà perdue, en décidant de la reprendre à un tour particulier. Enfin, ce système de sauvegarde rendrait possible un mode multi-joueur à distance, les joueurs pourraient s'envoyer par e-mail les parties après avoir joué leur coup afin que l'autre joueur puisse jouer à son tour.

- Créer un système de statistiques, permettant au joueur de consulter sa progression sous forme visuelle (diagrammes en ASCII), et un système de « succès ». (tel héros a été battu, une différences de score plus ou moins importante, un nombre de parties supérieur à X etc.)

- Mettre en place un didacticiel lors de la première partie ainsi qu'un mode apprentissage gérés par une Intelligence Artificielle, évaluant les performances du joueur et lui indiquant pourquoi un autre coup aurait été plus intéressant à jouer.

- Enfin, concevoir une interface graphique sur Qt, d'avantage orientée utilisateur, connectée au moteur du jeu en Prolog à l'aide d'un framework adapté.

Conclusion

La réalisation du jeu « Chicago Stock Exchange » nous aura permis de se familiariser autour du langage Prolog d'une manière originale et ludique (de la syntaxe pure au fonctionnement sous forme de liste en passant par les cuts), mais également d'améliorer notre gestion du travail en équipe.

De plus, sa conception nous aura surtout permis de s'interroger sur les façons de programmer et de « faire penser » un ordinateur afin qu'il puisse rivaliser avec un humain. Nous avons ainsi eu l'occasion de mettre en place notre première intelligence artificielle basée sur l'algorithme très puissant qu'est le MinMax, mais également d'en apercevoir ses faiblesses (telle que la complexité exponentielle).

Tout d'abord réticents à l'idée de programmer ce jeu et cette intelligence artificielle en Prolog, nous nous sommes rapidement rendus compte de la puissance de ce dernier et de la rapidité de développement qu'il offrait, comparativement au temps de développement en C/C++ qui aurait été bien plus grand.