IA02 - Projet

Rapport

Martin Salles, Hu Kewei

19/06/2013

Table des matières

I. Introduction

Ce projet réalisé dans l'UV IA02 a pour but de mettre en pratique les notions de programmation logique abordées en cours et en TD. Il s'agit de notre première utilisation du langage prolog.

Ce semestre, il s'agissait d'implémenter le jeu de plateau japonais Okiya. Le code prolog à rendre doit permettre de jouer entre humains, contre une machine, et de faire s'affronter deux machines.

Okiya se joue sur un plateau carré de seize cases au total, sur lesquelles sont posées aléatoirement seize tuiles. Une tuile est un couple sujet/végétation. Les joueurs doivent tour à tour prendre une tuile et la remplacer par un pion les identifiant. La seule contrainte est que la tuile choisie doit comporter le même sujet ou la même végétation que la tuile jouée par le joueur adverse au tour précédent. Un joueur gagne s'il aligne quatre de ses pions en ligne, colonne ou diagonale, s'il fait un carré ou si l'autre joueur ne peut plus jouer.

II. Représentation des connaissances

Une tuile est écrite sous la forme *[Sujet, Végétation]*. Par exemple [Oiseau, erable].

Les deux joueurs sont toujours identifiés comme *j1* et *j2*.

Le plateau de jeu est une liste de quatre lignes, chaque ligne étant aussi une liste de quatre tuiles. Cela permettra ensuite un affichage plus facile du plateau. Voici un exemple de plateau :

```
[[[oiseau, erable],[tanzaku,iris] ,[soleil,erable], [pluie,cerisier]], [[tanzaku,pin], [oiseau,pin], [pluie,erable], [soleil,iris]], [[pluie,iris], [pluie, pin], [soleil,cerisier], [oiseau,cerisier]], [[soleil,pin], [oiseau,iris], [tanzaku,cerisier], [tanzaku,erable]]]
```

Un coup est représenté par un couple joueur/tuile : *[j1, [oiseau,pin]]*

Pour jouer un coup, on demande au joueur d'entrer la tuile qu'il a choisie. Il devra donc entrer un couple *[sujet,végétation].* (Le point étant nécessaire pour marquer la fin de l'entrée avec le prédicat read).

III. Prédicats de jeu

A. Affichage du plateau

Nous profitons de la représentation du plateau en liste pour décomposer en prédicat l'affichage du plateau. Ainsi, le prédicat *affiche_plateau* appelle directement le prédicat *affiche_ligne*. Ce dernier permet d'écrire toutes les tuiles de la ligne à l'aide du prédicat *ecrire_tuiles*, puis d'afficher les lignes suivantes. Le prédicat *ecrire_tuile* fonctionne sur le même principe pour écrire successivement les quatre tuiles de la ligne.

```
ecrire_tuiles([T|[]]):-ecrire_tuile(T),!.
ecrire_tuiles([T|Q]):-ecrire_tuile(T), write(' | '), ecrire_tuiles(Q).
affiche_ligne([T|[]]):-ecrire_tuiles(T),!.
affiche_ligne([T|Q]):-ecrire_tuiles(T),nl, write('-----------------'), nl, affiche_ligne(Q).
affiche_plateau(L):-affiche_ligne(L),nl,nl,nl.
```

B. Génération du plateau

Pour générer le plateau, nous utilisons unifions une variable avec la liste des tuiles et sélectionnons la nième tuile de cette liste, n étant aléatoire. Nous construisons le plateau de la même manière qu'il est affiché, c'est-à-dire ligne par ligne, puis tuile par tuile dans une ligne.

C. Coup possible

Pour déterminer si un coup est possible, nous vérifions que la tuile que l'on veut jouer est dans le plateau, et qu'elle correspond à la tuile jouée au tour précédent (même sujet ou même végétation). Au premier tour, il suffit de vérifier qu'elle est dans le plateau.

```
coup_possible(Plateau,[],[_,T]):-in_plateau(Plateau,T).
coup_possible(Plateau,[T1,T2],[_,[T1,Y]]):-in_plateau(Plateau,[T1,Y]).
coup_possible(Plateau,[T1,T2],[_,[X,T2]]):-in_plateau(Plateau,[X,T2]).
```

D. Boucle de jeu

Notre boucle de jeu consiste à faire jouer tour à tour les joueurs en s'assurant que les coups qu'ils jouent sont valides. A chaque tour, le plateau est mis à jour, on vérifie s'il y a une condition de victoire, puis on rappelle la boucle de jeu pour l'adversaire en indiquant la tuile qui vient d'être jouée.

```
boucle_jeu(Plateau, Joueur, Vainqueur,TuilePrecedente):-
	affiche_plateau(Plateau),
	write('C''est au tour du joueur '),write(Joueur),write(' de jouer :'),nl,
	read(Tuile),
	Coup = [Joueur,Tuile],
	(
	coup_possible(Plateau,TuilePrecedente,Coup)->
		jouer_coup(Plateau, Coup, NouveauPlateau),
		(
			victoire(Joueur, Tuile, NouveauPlateau)->
				Vainqueur = Joueur,
				affiche_plateau(NouveauPlateau)
			;
				change_joueur(Joueur, NouveauJoueur),
				boucle_jeu(NouveauPlateau, NouveauJoueur, Vainqueur,Tuile)
		)
	;
		nl,write('Tuile incorrecte'),nl,nl,
		boucle_jeu(Plateau, Joueur, Vainqueur,TuilePrecedente)
	).

humainhumain(Vainqueur):-
	plateau_depart(Plateau),
	boucle_jeu(Plateau, j1, Vainqueur,[]),
	write('Le vainqueur est : '), write(Vainqueur),!.
```

Les boucles de jeu humain vs IA et IA vs IA fonctionnent sur le même principe en remplaçant l'entrée d'une tuile par le joueur par la sélection du coup le plus pertinent.

IV. Intelligence Articifielle

Le jeu de l'IA se base principalement sur l'algorithme du minimax, qui est le plus utilisé pour des jeux de ce type (sans hasard et où on connaît toute la situation à un instant t).

A. Minimax

L'objectif de cet algorithme est de parcourir tous les états possibles en fonction des coups que les joueurs pourraient jouer dans les tours suivants. Nous avons décidé d'effectuer cette recherche à une profondeur de 5. Cela nous permet de déterminer un meilleur coup avec une très bonne précision, tout en évitant une erreur de dépassement de pile.

L'algorithme consulte tous les coups possibles pour un joueur et sélectionne celui qu'il a le plus intérêt à jouer. Il construit donc une arborescence et donne des valeurs à tous les plateaux. Pour un coup à jouer par le joueur, on sélectionne le plateau le plus favorable parmi les plateaux résultants des coups possibles. A l'inverse, pour l'adversaire, on sélectionnera celui qui a l'évaluation la plus basse. On part donc du principe que l'adversaire joue au mieux également.

```
meilleur_coup(Plateau, TuilePrecedente, Joueur, JoueurQuiJoue, Profondeur, MC, MV):-
        coups_possibles(Plateau, TuilePrecedente, ListeCoups),
        Pr2 is Profondeur-1,
        minmax(Plateau, TuilePrecedente, Joueur, JoueurQuiJoue, Pr2, ListeCoups, MC, MV).

choisir_min_max(Joueur, JoueurQuiJoue, MC1, MV1, MC2, MV2, MC, MV):-
        Joueur == JoueurQuiJoue ->
                max(MV1, MV2, MC1, MC2, MV, MC)
        ;
                min(MV1, MV2, MC1, MC2, MV, MC).

minmax(Plateau,_,Joueur,JoueurQuiJoue,Profondeur,[C1],C1,MV):-!,
        jouer_coup(Plateau, [Joueur,C1], NouveauPlateau),
        (
        feuille(NouveauPlateau,Profondeur,C1,Joueur) -> %(si gagnant, perdant ou profondeur=0)
                evaluer(NouveauPlateau, Joueur,JoueurQuiJoue,C1,MV)
        ; %(sinon)
                change_joueur(Joueur, Adversaire),
                meilleur_coup(NouveauPlateau, C1, Adversaire,JoueurQuiJoue, Profondeur,_, MV)
        ).

minmax(Plateau, TuilePrecedente, Joueur, JoueurQuiJoue, Profondeur, [C1|Autrescoups], MC, MV):-
        jouer_coup(Plateau, [Joueur,C1], NouveauPlateau),
        (feuille(NouveauPlateau,Profondeur,C1,Joueur) -> %(si gagnant, perdant ou profondeur=0)
                evaluer(NouveauPlateau, Joueur,JoueurQuiJoue,C1,V1)
        ; %(sinon)
                change_joueur(Joueur, Adversaire),
                meilleur_coup(NouveauPlateau, C1, Adversaire,JoueurQuiJoue, Profondeur, _, V1)
        ),
        minmax(Plateau, TuilePrecedente, Joueur, JoueurQuiJoue, Profondeur, Autrescoups, C2, V2),
        choisir_min_max(Joueur, JoueurQuiJoue, C1, V1, C2, V2, MC, MV).
```

B. Evaluation d'un plateau

On évalue un plateau s'il est feuille dans l'arborescence des plateaux successifs. Une feuille est un plateau gagnant, ou un plateau à la profondeur maximale.

```
feuille(_,0,_,_):-!.
feuille(Plateau,_,TuilePrecedente,Joueur):-victoire(Joueur,TuilePrecedente,Plateau).
```

Pour ces plateaux, on cherche à calculer directement dans quelle mesure il est favorable au joueur. Pour cela, on regarde tout d'abord si le plateau est gagnant pour le joueur ou son adversaire, auquel cas on lui attribue une valeur considérée comme infinie : 999 ou -999.

```
evaluer(Plateau,Joueur,Joueur,TuilePrecedente,999):-victoire(Joueur,TuilePrecedente,Plateau),!.
evaluer(Plateau,Joueur,JoueurQuiJoue,TuilePrecedente,-999):-
        change_joueur(Joueur,JoueurQuiJoue),victoire(Joueur,TuilePrecedente,Plateau),!.
```

Dans les autres cas, nous avons décidé de calculer le nombre de lignes, colonnes, diagonales ou carrés qui peuvent potentiellement devenir gagnants pour le joueur. On soustrait ensuite le nombre de situations qui pourraient devenir gagnantes pour l'adversaire. On obtient alors un nombre qui est d'autant plus grand que le joueur a plus de situations gagnantes que l'adversaire.

```
evaluer(Plateau, Joueur, _,_, Valeur):-
        plateau_factice(Plateau, Joueur, PlateauFactice1),
        findall(X, compte_reussi(Joueur, PlateauFactice1, X), ListeJoueur),
        length(ListeJoueur, SommeJoueur),
        change_joueur(Joueur, AutreJoueur),
        plateau_factice(Plateau, AutreJoueur, PlateauFactice2),
        findall(Y, compte_reussi(AutreJoueur, PlateauFactice2, Y), ListeAutreJoueur),
        length(ListeAutreJoueur, SommeAutreJoueur),
        Valeur is SommeJoueur - SommeAutreJoueur.
```

V. Difficultés rencontrées

La majeure difficulté du projet était que c'était la première fois que nous devions réaliser un travail en prolog. Le mode de raisonnement étant totalement différent de ce que nous avions rencontré auparavant, il nous a fallu prendre le temps de nous habituer pour travailler efficacement.

Nous avons notamment eu du mal à débugger notre programme. Lorsqu'une erreur que nous ne comprenions pas survenait, nous essayions de lancer un à un les prédicats avec les valeurs du programme pour les variables. Cependant, nous avons parfois du utiliser la commande *trace*, ce qui s'est révélé très fastidieux, surtout lorsque des appels très nombreux à plusieurs prédicats sont réalisés.

Le temps a également été une contrainte forte, puisque le projet a démarré tard dans le semestre.

VI. Améliorations possibles

Le jeu implémenté n'étant disponible que dans une console prolog, il n'est pas accessible aux non-informaticiens. Il pourrait donc être intéressant de développer une interface graphique, qui permettrait également un affichage du plateau plus visuel et une interface plus ergonomique pour jouer un coup.

En ce qui concerne l'intelligence artificielle, on pourrait utiliser un algorithme comme l'élagage alpha beta. Cela permettrait de ne pas développer des parties inutiles de l'arborescence des coups possibles et des plateaux successifs associés. Ainsi, la recherche du meilleur coup pour un joueur IA serait plus performante puisqu'on pourrait l'utiliser jusqu'à une profondeur plus grande.

De même, on pourrait améliorer la fonction d'évaluation d'un plateau de jeu. En effet, on plutôt que de simplement lister les situations qui pourraient devenir victorieuses, on pourrait aussi regarder de quelle manière. Par exemple, si en jouant une tuile, on pourrait avancer pour compléter une ligne, une colonne, un carré et une diagonale en même temps, la situation est plus profitable que si l'ajout d'une tuile ne nous fait avancer que pour une ligne par exemple. Par ailleurs, cette fonction d'évaluation ne prend pas en compte les possibilités de bloquer l'adversaire.

VII. Conclusion

Cette première expérience prolog a été très enrichissante car nous sommes partis de zéro pour réaliser un jeu qui est fonctionnel. Notre programme est bien évidemment perfectible mais nous sommes satisfaits du rendu.

Ce projet nous a permis de nous rendre compte de l'intérêt de la programmation logique pour la résolution d'un certain nombre de problèmes. Plusieurs de nos prédicats sont beaucoup plus courts que ce qui aurait été nécessaires dans d'autres langages de programmation que nous connaissions déjà. La prise en main a été un peu compliquée au début mais nous avons petit à petit réussi à réfléchir pour programmer en prolog et à sortir d'un mode de réflexion « procédural ».