Lefevre Victor
Gomez Guillaume

17/06/13

RAPPORT de IA02
Projet Prolog :
Okiya

Table des matières

1) Introduction

Le but de ce projet était multiple, dans un premier temps il nous a permit d'approfondir et de mettre en pratique les connaissances du langage de programmation *Prolog*. Puis de concevoir une véritable intelligence artificielle capable d'être efficace dans les décisions prises, tout en étant rapide.

2) Partie graphique

2.1) Structure de données

Le plateau de jeu est représenté par une liste de liste, qui correspond à une matrice. La structure étant relativement classique dans ce genre de projet, la compréhension de la structure nous parut intuitive. Toute la complexité résidait dans la modification de cette structure au cours du jeu.

Nous avons choisi de convertir les tuiles en nombres et inversement pour réaliser les traitements du jeu.

```
% liste des cases à transformer
case([oiseau,erable],1).
case([oiseau,iris],2).
case([oiseau,pin],3).
case([oiseau,cerisier],4).
case([tanzaku,erable],5).
case([tanzaku,iris],6).
case([tanzaku,pin],7).
case([tanzaku,cerisier],8).
case([pluie,erable],9).
case([pluie,iris],10).
case([pluie,pin],11).
case([pluie,cerisier],12).
case([soleil,erable],13).
case([soleil,iris],14).
case([soleil,pin],15).
case([soleil,cerisier],16).
```

2.2) Affichage du plateau

Concernant l'affichage du plateau, nous avons décidé de diviser les taches en plusieurs prédicats pour rendre le problème le plus aisé possible. Ainsi avons nous un prédicat général affiche_plateau qui affiche le plateau complet. Ce prédicat appelle un prédicat qui permet l'affichage des lignes, qui lui même appelle un prédicat pour afficher un élément du plateau.

```
% affiche le plateau sous forme de tableau
affiche_plateau([]) :- nl.
affiche_plateau([T|Q]) :-
%write('-----------------------------------------'),nl,
        affiche_ligne(T),
        affiche_plateau(Q),!.

%Parcours une ligne de la matrice 4*4
affiche_ligne([A,B,C,D]) :-
   write('| '),
        affiche_element(A),
        affiche_element(B),
        affiche_element(C),
        affiche_element(D),
        nl,write('-----------------------------------------'),nl.

affiche_element([A,B]) :-
        trad(A),
        write(' - '),
        trad(B),
        print('  |  ').

% Permet de sauvegarder l'allure generale de l'affichage du plateau lorsqu'un joueur prend une case
affiche_element(E) :-
        print('  '),
        trad(E),
        print('   |  ').
```

2.3) Génération du plateau de départ

Pour la génération du plateau, nous sommes partis dans l'idée de réaliser un tableau qui respectait la structure de données (c'est à dire un liste qui se compose de liste). On à donc choisi de créer un tableau de 16 éléments allant de 1 à 16, puis de randomiser les occurrences de ce tableau. De récupérer un à un les éléments du tableau pour les remplacer par les tuiles du jeu. Enfin, la tableau est transformé d'une seule dimension vers une matrice 4*4.

```
plateau_depart(R) :-
write('-----------------------------------'),nl,
        list_init(L),
        aleat(L,T),
        convert_number_to_case(T,Q),
        list_to_plateau(Q,R).

% Itere sur les lignes et les colonnes pour choisir un nombre al é atoirement
aleat(L,[T|Q]) :-
        aleat1(L,T,H),!,
        aleat(H,Q).

aleat(_,[]).

aleat1(L,R,H) :-
        length(L,Y),
        W is Y + 1,
        random(1,W,X),
        nth(X,L,R),
        delete(L,R,H).

% Pr é dicat qui permet de generer un tableau de 16 cases
list_init([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16]).

% convertit un tableau en matrice
list_to_plateau([A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P],[[A,B,C,D],[E,F,G,H],[I,J,K,L],[M,N,O,P]]).

% convertit un nombre en chaine
convert_number_to_case([T|Q],[R|S]) :-
        case(R,T),!,
        convert_number_to_case(Q,S).

convert_number_to_case([],[]).
```

```
% liste des cases à transformer
case([oiseau,erable],1).
case([oiseau,iris],2).
case([oiseau,pin],3).
case([oiseau,cerisier],4).
case([tanzaku,erable],5).
…
```

3) Interactions utilisateur

L'interaction avec l'utilisateur se veut simpliste, en effet elle se contente de demander au joueur l'abscisse et l'ordonné de la tuile qu'il souhaite jouer.

Le plus difficile fut la gestion du coup. Il fallait vérifier si le coup entré par le joueur était conforme au règles du jeu. On à donc implémenter le prédicat coup_possible qui cherche tout les coups possibles pour un plateau à l'instant *t,* et vérifie si le coup jouer par le joueur appartient bien à cette liste de coup.

Une liste des coups possibles est proposé au joueur pour l'aider dans sa décision en utilisant le prédicat coup_possible.

```
%renvoit vrai ou faux si le coup passé en parametre est possible ou non
% P plateau en parametre
% [T|Q] La tuile jouée par le joueur precedent
% [J(X,Y)] Coup que l'on souhaite tester
coup_possible(_,[],[_,(_,_)]).
coup_possible(P,[T,Q],[_,(X,Y)]) :-
        nth(X,P,V),
        nth(Y,V,U),
        match([T,Q],U),!.

%Renvoit la liste des coups possibles
% voir coup_possible2
coups_possibles(P,[],L,J) :-
        setof((X,Y), coups_possibles2(P,[],[J,(X,Y)]), L),!.

coups_possibles(P,[T,Q],L,J) :-
        setof([ J,(X,Y) ], coups_possibles2(P,[T,Q],[J,(X,Y)]), L).
%teste si les indices sont cohérent ( entre 1 et 4)
coups_possibles2(P,[],[J,(X,Y)]) :-
        member(X,[1,2,3,4]),
        member(Y,[1,2,3,4]).
```

```
coups_possibles2(P,[T,Q],[J,(X,Y)]) :-
        nth(X,P,V),
        nth(Y,V,U),
        match([T,Q],U).
% P le plateau
% [J,(X,Y)] le coup à réaliser par un joueur(1 ou 2)
% R le plateau modifié apres le coup
% U la tuille jouée
jouer_coup(P,[J,(X,Y)],R,U) :-
        nth(X,P,V),
        nth(Y,V,U),
        select(U,V,W),
        insert(J,W,Y,Z),
        select(V,P,A),
        insert(Z,A,X,R),!.
```

4) Le menu

Le prédicat menu permet à l'utilisateur de choisir quel mode de jeu il souhaité utilisé. Dans le cas ou la valeur entrée est incorrecte, le prédicat renvoie un message d'erreur.

```
--------------------------------
1 : Mode 2 Joueurs
2 : Mode 1 Joueur
3 : Mode 2 Joueurs IA version 1
4 : Mode 2 Joueurs IA version 2
Entrer votre choix --> 1.


Joueur j1
---------+---------+---------+---------+
| O - E  |  P - I  |  O - C  |  S - I  |
---------+---------+---------+---------+
---------+---------+---------+---------+
| P - P  |  T - C  |  T - I  |  S - C  |
---------+---------+---------+---------+
---------+---------+---------+---------+
| S - E  |  P - C  |  O - P  |  S - P  |
---------+---------+---------+---------+
---------+---------+---------+---------+
| T - P  |  T - E  |  O - I  |  P - E  |
---------+---------+---------+---------+

Coup Possible -->[[j1,(1,1)],[j1,(1,2)],[j1,(1,3)],[j1,(1,4)],[j1,(2,1)],[j1,(2,2)],[j1,(2,3)],[j1,(2,4)]
valeur x
2.
valeur y
```

Illustration 1: fonctionnement du menu

5) L'intelligence artificielle

Lors de l'élaboration de l'intelligence artificielle, nous avons développé 2 algorithmes de programmation. Le premier est limité car, il n'est capable de rechercher d'en profondeur 3. Quant au second, il est issu de l'algorithme Minimax vu en cours ainsi qu'en travaux pratiques.

5.1) Premier algorithme

L'algorithme est construit sur trois fonctions, qui vont respectivement réaliser le traitement pour une hauteur donnée de l'arbre de recherche.

```
%niveau 1 de l'arbre
meilleur_coup(P,[],[J,(X,Y)]) :-
        coups_possibles(P,[],L,J),!,
        length(L,Z),
        W is Z + 1,
        random(1,W,N),
        nth(N,L,(X,Y)).

meilleur_coup(P,[T,Q],[J,(X,Y)]) :-
        coups_possibles(P,[T,Q],L,J),!,
        meilleur_coup2(P,L,[J,(X,Y)],R),
        max1(R,W),
        nth(Z,R,W),
        nth(Z,L,[J,(X,Y)]).
%niveau 2 de l'arbre
meilleur_coup2(_,[],[_,(_,_)],[]).

meilleur_coup2(P,[T|Q],[J,(X,Y)],[R|S]) :-
        jouer_coup(P,T,Psuiv,Tsuiv),!,
        joueur_suivant(J,Jsuiv),
        (continuer(Psuiv,Tsuiv,Jsuiv),coups_possibles(Psuiv,Tsuiv,L,Jsuiv) ->
                meilleur_coup3(Psuiv,L,[Jsuiv,(X,Y)],A),
                min1(A,R),
                meilleur_coup2(P,Q,[J,(X,Y)],S);
                print('egalite'),nl,affiche_plateau(PSuiv),break).

%niveau 3 de l'arbre
meilleur_coup3(_,[],[_,(_,_)],[]).

meilleur_coup3(P,[T|Q],[J,(X,Y)],[R|S]) :-
```

```
        jouer_coup(P,T,Psuiv,Tsuiv),!,
        heuristique(Psuiv,J,R),
        meilleur_coup3(P,Q,[J,(X,Y)], S).
```

Le problème majeur de cette algorithme est qu'il est peu réutilisable pour d'autre problème. D'ailleurs il faudrait changer l'algorithme si on voulait une recherche de niveau 4 dans l'arbre de recherche.

A l'instar de l'algorithme Minimax, le cœur de la décision prise par l'algorithme dépend d'une fonction d'évaluation qui va décider du meilleurs coup dans l'arbre des coup possibles.

Cette fonction se nomme ici « heuristique »

```
evaluer([],_,0,T,T).

evaluer(Plateau,Joueur,Resultat,T,T):-
        joueur_suivant(J,Jsuiv),
        %on remplit le tableau avec le joueur pour
        remplir_plateau(P,J,Jsuiv,PRempli),
        %on recupere le nombre de diagonales pour ce plateau
        (findall(X1,diagonale(PRempli,J,X1), L1) ->
                length(L1,Z1);
                Z1 is 0),
        %analague
        (findall(X2,antidiagonale(PRempli,J,X2), L2) ->
                length(L2,Z2);
                Z2 is 0),
        %idem
        (findall(X3,ligne(PRempli,J,X3), L3) ->
                length(L3,Z3);
                Z3 is 0),
        (findall(X4,antidiagonale(PRempli,J,X4), L4) ->
                length(L4,Z4);
                Z4 is 0),
       (findall(X5,carre(PRempli,J,X5), L5) ->
                length(L5,Z5);
                Z5 is 0),
        % on somme le tout
        R is Z1 + Z2 + Z3 + Z4 + Z5.
```

5.2) Second algorithme

Ce second algorithme est basé de Minimax. Elle se décompose en trois fonctions :

- meilleur_coupIA:

  Retourne le meilleur coup parmi la liste des coup possibles pour un plateau t,

- min_max :

  Génère un arbre d'état des coup possibles,

- choisirMinMax :

  En fonction de notre position dans l'arbre, on retourne le coup à valeur minimal ou maximal.

```
%on simule un coup, puis on r é cup è re le meilleur coup parmi les coup simul é es
min_max(Plateau,DerT,DerC,Prof,[[Joueur,(X,Y)]|ResteCouP],MeCoup,MeVal) :-
        jouer_coup(Plateau,[Joueur,(X,Y)],Nouv_Plateau,NouvTuile),
        Prof2 is Prof - 1,
        joueur_suivant(Joueur,JSuiv),
        meilleur_coupIA(Nouv_Plateau,NouvTuile,[JSuiv,(X,Y)],Prof2,MeCoup1,MeVal1),
        min_max(Plateau,DerT,[Joueur,(X,Y)],Prof,ResteCouP,MeCoup2,MeVal2),
        choisirMinMax(DerC,MeVal1,[Joueur,(X,Y)],MeVal2,MeCoup,MeVal,Prof2).

%on evalue un coup simul é  si on atteint une feuille, puis on resimule un coup avec min_max
meilleur_coupIA(Plateau,DerT,[Joueur,(X,Y)],Prof,MeCoup,MeVal) :-
        coups_possibles(Plateau,DerT,ListCp,Joueur),
        (feuille(ListCp,Prof) ->
                evaluer(Plateau,Joueur,MeVal,MeCoup,[Joueur,(X,Y)])
        ;
                min_max(Plateau,DerT,[],Prof,ListCp,MeCoup,MeVal)
        ).
%on choisi le meilleur coup
choisirMinMax([],_,MeCoup2,_,MeCoup2,_,_).
choisirMinMax(MeCoup1,MeVal1,_,[],MeCoup1,MeVal1,_).
choisirMinMax(_,[],MeCoup2,MeVal2,MeCoup2,MeVal2,_).

choisirMinMax(MeCoup1,MeVal1,MeCoup2,MeVal2,MeCoup,MeVal,Prof):-
        %print(MeCoup1),
        (Prof mod 2 =:= 1 ->
                max(MeCoup1,MeVal1,MeCoup2,MeVal2,MeCoup,MeVal)
        ;
                min(MeCoup1,MeVal1,MeCoup2,MeVal2,MeCoup,MeVal)
        ).
```

6) Amélioration possible

Premièrement, nous aurions pu améliorer l'intelligence artificielle dans sa globalité. Cette amélioration aurait pu être faite par la modification du prédicat « evaluer ». Cela permettrait d'avoir des prises de décision plus spécifique. En effet dans l'état actuel, le prédicat choisi le premier coup parmi une liste de coup à valeur égale.

De plus nous aurions pu implémenter l'élagage alpha-beta pour augmenter la rapidité de l'algorithme minimax. Malheureusement, nous avons manqué de temps.

Enfin, il aurait été intéressant de formaliser certains prédicats, pour qu'il puisse fonctionner dans le cas général. On peut citer les prédicats suivants qui sont confrontés à ce problème : « carre », »diagonale », « ligne », « colonne »...

7) Conclusion

A travers ce projet, nous avons pu voir la puissance du langage *prolog*, qui nous a permit de développer un jeu en seulement un centaine de ligne, la ou d'autre langages nous aurait demandé un investissement plus important en matière de programmation. De plus on a pu approfondir ce langage qui possède un paradigme de programmation si particulier.

Enfin, on a pu véritablement développer une véritable intelligence artificielle, qui est capable de jouer à « Okiya » mais aussi à un grande partie des jeux de plateau deux joueurs (avec quelques modifications)