Universidade de Évora

Inteligência Artificial

Relatório do 3º Trabalho Prático

Autor:
Pedro Nunes, 31240
Tiago Martinho, 35735

Docente:
Paulo Quaresma

Abril de 2018

Índice

1 Jogos com informação completa determinísticos

Definição do problema

Pretende-se representar o jogo "Três em Linha"como um problema de pesquisa no espaço de estados. O jogo é realizado sobre um tabuleiro de 5 colunas e 4 linhas. Em cada jogada, os jogadores colocam uma ficha da sua cor numa coluna e esta cai até á primeira casa disponível naquela coluna. O que conseguir colocar 3 fichas da mesma cor seguidas na horizontal, vertical ou diagonal ganha. Se ninguém conseguir, a partida termina em empate.

Neste trabalho poderá escolher entre jogar como jogador ou colocar um bot que joga sozinho, para isso basta escrever o predicado `joga_jogador(Coluna)` para jogador ou `joga_bot()` para o bot.

Estrutura de dados

Foi utilizada uma lista com listas dentro, a lista principal representa as colunas e cada lista dentro dessa representa as linhas.

```
:- dynamic(estado_inicial/1).
estado_inicial([[v,v,v,x],[v,v,o,x],[v,o,x,o],[v,o,x,x],[v,x,o,o]]).
```

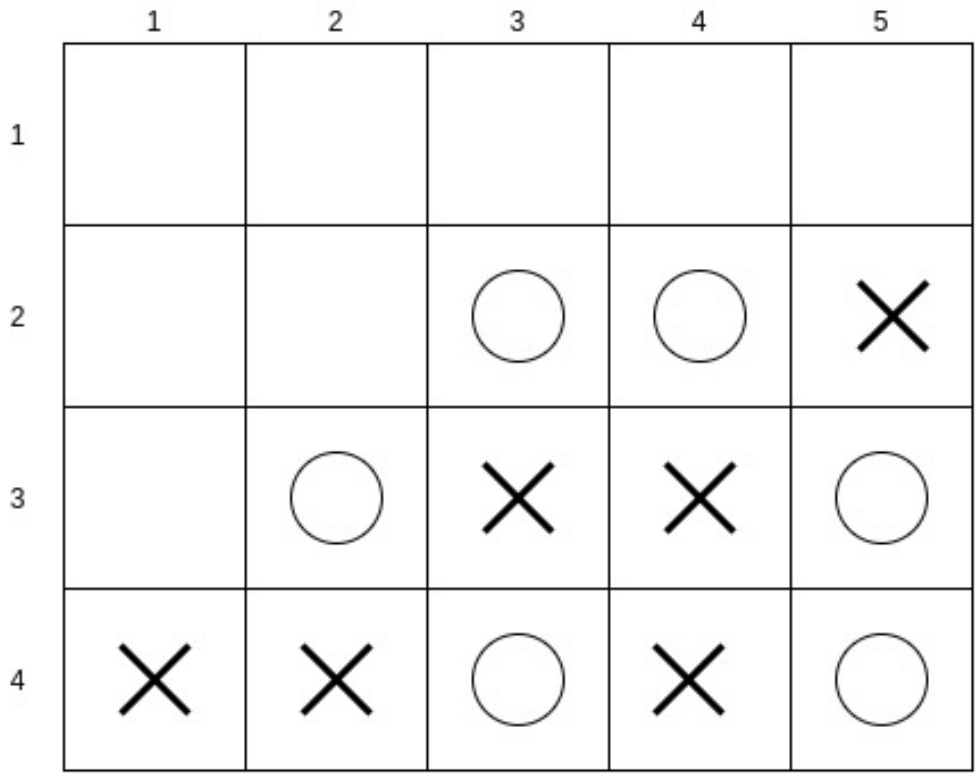

Figura 1 - Estado inicial do jogo.

Predicado Terminal

```
terminal(G) :- linhas(G,_).
terminal(G) :- colunas(G,_).
terminal(G) :- diagonal(G,_).
terminal(G) :- cheio(G).
```

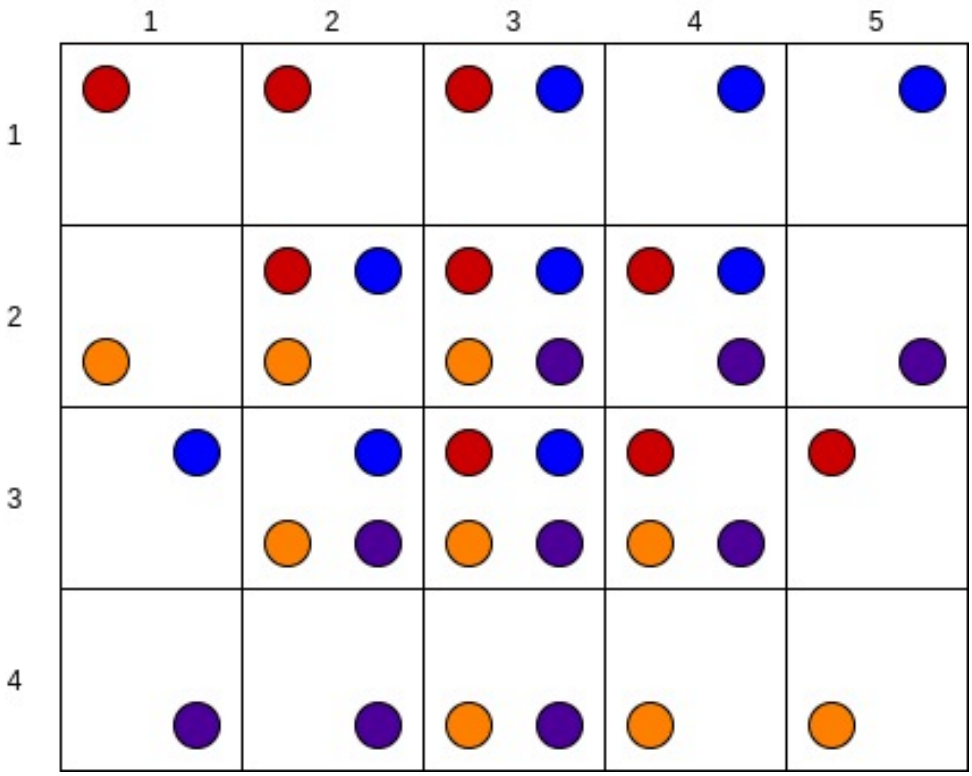

Figura 2 - Diagrama das diagonais terminais possíveis

Defina uma função de utilidade que, para um estado terminal, deve retornar o valor do estado (ex: -1 perde, 0 empata, 1 ganha)

```
valor(G, 10) :- linhas(G,x).
valor(G, 10) :- colunas(G,x).
valor(G, 10) :- diagonal(G,x).
valor(G, -10) :- linhas(G,o).
valor(G, -10) :- colunas(G,o).
valor(G, -10) :- diagonal(G,o).
valor(_, 0).
```

Use a implementação da pesquisa minimax para escolher a melhor jogada num estado

```
:-dynamic(visitados/1).
visitados(0).

joga(Op) :-
        estado_inicial(Ei),
        minimax_decidir(Ei,Op),
        visitados(V),
        write('Bot ->'), write(Op),nl,
    write('Nos visitados: '), write(V), nl.

% decide qual e' a melhor jogada num estado do jogo
% minimax_decidir(Estado, MelhorJogada)

% se e' estado terminal nao ha jogada
minimax_decidir(Ei,terminou) :-
        terminal(Ei).

% Para cada estado sucessor de Ei calcula o valor minimax do estado
% Opf e' o operador (jogada) que tem maior valor
% Nota: assume que o jogador e' o "x"
minimax_decidir(Ei,Opf) :-
        findall(Vc-Op, (oper(Ei,x,Op,Es), minimax_valor(Es,Vc,1)), L),
        escolhe_max(L,Opf).

% se um estado e' terminal o valor e' dado pela funcao de utilidade
% Nota: assume que o jogador e' o "x"
minimax_valor(Ei,Val,_) :-
    retract(visitados(V)),
    V1 is V + 1,
    asserta(visitados(V1)),
        terminal(Ei),
        valor(Ei,Val).

%Se o estado nao e' terminal o valor e':
% -se a profundidade e' par, o maior valor dos sucessores de Ei
% -se aprofundidade e' impar o menor valor dos sucessores de Ei
minimax_valor(Ei,Val,P) :-
        P1 is P+1, jogador(P1,J),
        findall(Val1, (oper(Ei,J,_,Es), minimax_valor(Es,Val1,P1)), V),
        seleciona_valor(V,P,Val).


% jogador "x" nas jogadas impares e jogador "o" nas jogadas pares
jogador(P, o) :- X is P mod 2, X = 0.
jogador(P, x) :- X is P mod 2, X = 1.
```

```
% Se a profundidade (P) e' par, retorna em Val o maximo de V
seleciona_valor(V,P,Val) :-
        X is P mod 2, X=0,!,
        maximo(V,Val).

% Senao retorna em Val o minimo de V
seleciona_valor(V,_,Val):- minimo(V,Val).

%% Predicados auxiliares

escolhe_max([A|R],Val):- escolhe_max(R,A,Val).

escolhe_max([],_-Op,Op).
escolhe_max([A-_|R],X-Op,Val) :- A < X,!, escolhe_max(R,X-Op,Val).
escolhe_max([A|R],_,Val):- escolhe_max(R,A,Val).


maximo([A|R],Val):- maximo(R,A,Val).

maximo([],A,A).
maximo([A|R],X,Val):- A < X,!, maximo(R,X,Val).
maximo([A|R],_,Val):- maximo(R,A,Val).

minimo([A|R],Val):- minimo(R,A,Val).

minimo([],A,A).
minimo([A|R],X,Val):- A > X,!, minimo(R,X,Val).
minimo([A|R],_,Val):- minimo(R,A,Val).
```

Implemente a pesquisa Minimax com corte alfa-beta e compare os resultados

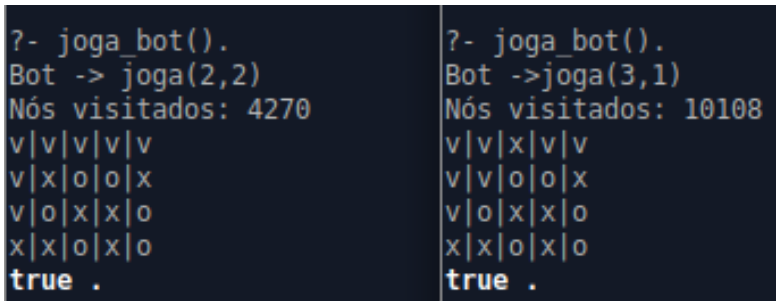

Figura 3 - Comparação dos resultados entre Minimax com corte Alfa-Beta e Minimax (respectivamente).

Como podemos observar a pesquisa Minimax com corte Alfa-Beta visita menos nós, 4270, contra 10108 com pesquisa Minimax. Quanto á jogada, com a pesquisa Alfa-Beta, esta é muito melhor em comparação á jogada feita pela pesquisa Minimax.

```
:-dynamic(visitados/1).
visitados(0).

joga(Op) :-
        estado_inicial(Ei),
        alfabeta(Ei,Op),
        visitados(V),
        write('Bot -> '), write(Op),nl,
    write('Nos visitados: '), write(V), nl.

% decide qual e' a melhor jogada num estado do jogo
% alfabeta(Estado, MelhorJogada)

% se e' estado terminal nao ha jogada
alfabeta(Ei,terminou) :-
    retract(visitados(V)),
    V1 is V + 1,
    asserta(visitados(V1)),
    terminal(Ei).

% Nota: assume que o jogador e' o "x"
alfabeta(Ei,Opf) :-
        findall(Vc-Op, (oper(Ei,x,Op,Es),
            alfabeta_min(Es,Vc,1,-10000,10000)), L),
            escolhe_max(L,Opf).

% se um estado e' terminal o valor e' dado pela funcao de utilidade
```

```
% Nota: assume que o jogador e' o "x"
alfabeta_min(Ei,Val,_,_,_) :-
        retract(visitados(V)),
    V1 is V + 1,
    asserta(visitados(V1)),
        terminal(Ei),
        valor(Ei,Val), !.

alfabeta_min(Ei,Val,P,Alfa,Beta) :-
        P1 is P+1, jogador(P1,J),
        V is 10000,
        findall(Es, oper(Ei,J,_,Es), L),
        processa_lista_min(L, P1, V, Alfa, Beta, Val), !.

processa_lista_min([], _, V, _, _, V).
processa_lista_min([H|T], P, V, A, B, V1) :-
        alfabeta_max(H, V2, P, -10000, 10000),
        min(V, V2, V3),
        (V3 < A, V1 is V3; min(B, V3, B1),
            processa_lista_min(T, P, V3, A, B1, V1)).

min(A,B,A) :- A < B, !.
min(_, B, B).

alfabeta_max(Ei,Val,_,_,_) :-
        terminal(Ei),
        valor(Ei,Val), !.

alfabeta_max(Ei,Val,P,Alfa,Beta) :-
        P1 is P+1, jogador(P1,J),
        V is -10000,
        findall(Es, oper(Ei,J,_,Es), L),
        processa_lista_max(L, P1, V, Alfa, Beta, Val), !.

processa_lista_max([], _, V, _, _, V).
processa_lista_max([H|T], P, V, A, B, V1) :-
        alfabeta_min(H, V2, P, -10000, 10000),
        max(V, V2, V3),
        (V3 >= B, V1 is V3; max(A, V3, A1),
            processa_lista_max(T, P, V3, A1, B, V1)).

max(A,B,B) :- A < B, !.
max(A, _, A).

% jogador "x" nas jogadas impares e jogador "o" nas jogadas pares
jogador(P, o) :- X is P mod 2, X = 0.
jogador(P, x) :- X is P mod 2, X = 1.
```

```
% Se a profundidade (P) e' par, retorna em Val o maximo de V
seleciona_valor(V,P,Val) :-
        X is P mod 2, X=0,!,
        maximo(V,Val).

% Senao retorna em Val o minimo de V
seleciona_valor(V,_,Val):- minimo(V,Val).

%% Predicados auxiliares

escolhe_max([A|R],Val):- escolhe_max(R,A,Val).

escolhe_max([],_-Op,Op).
escolhe_max([A-_|R],X-Op,Val) :- A < X,!, escolhe_max(R,X-Op,Val).
escolhe_max([A|R],_,Val):- escolhe_max(R,A,Val).


maximo([A|R],Val):- maximo(R,A,Val).

maximo([],A,A).
maximo([A|R],X,Val):- A < X,!, maximo(R,X,Val).
maximo([A|R],_,Val):- maximo(R,A,Val).

minimo([A|R],Val):- minimo(R,A,Val).

minimo([],A,A).
minimo([A|R],X,Val):- A > X,!, minimo(R,X,Val).
minimo([A|R],_,Val):- minimo(R,A,Val).
```

Implemente um agente inteligente que joga o jogo "Três em Linha".

```
joga_bot() :-
        joga(joga(X,Y)),
        retract(estado_inicial(Ei)),
        joga_vazio(Ei, x, X, Y, En),
        asserta(estado_inicial(En)),
        imprime_tabuleiro(En).
```