UNIVERSIDADE DE ÉVORA

Inteligência Artificial
2015/2016

Jogos de dois jogadores- Jogos com informação completa determinísticos

Docente:
Irene Pimenta Rodrigues

Discentes:
Pedro Jacob - 27421
André Figueira - 31626

Introdução

Neste problema que nos foi apresentado vamos tentar explorar por alto o campo da teoria da decisão, em particular maneiras de minimizar a perda máxima possível.

Para tal, vamos usar os conhecimentos obtidos nas aulas de Inteligencia Artificial, e unifica-los com a nossa capacidade de resolução de problemas e domínio sobre programação lógica (prolog).

Vamos neste trabalho apresentar dois casos parecidos, mas com diferenças suficientes para serem considerados separadamente: O jogo do galo, e o 4 em linha.

1. Jogo do Galo

1.1. Escolha uma estrutura de dados para representar os estados.

A escolha para a estrutura de dados para este problema foi escolhido um tuplo com dois elementos. O primeiro elemento é uma lista que contém as posições do tabuleiro 3x3, e o segundo elemento contém a informação de quem jogou.

```
estado_inicial(
                ([(p(1,1), _),(p(1,2), _),(p(1,3), _),
                (p(2,1), _),(p(2,2), _),(p(2,3), _),
                (p(3,1), _),(p(3,2), _),(p(3,3), _)],o)
            ).
```

1.2 Estado terminal

O jogo terminal se houver uma sequência de Xs ou Os em linha, ou coluna, ou diagonal. Ou então o jogo pode terminar se houver um empate, isto é, se o tabuleiro estiver todo preenchido.

```
% ########################################################################################

% terminal
terminal((EstadoActual, _)):-
    check_victoria(EstadoActual);
    empate(EstadoActual).

% ########################################################################################

check_victoria(EstadoActual):-
    (victoria_linhas(EstadoActual);
        victoria_colunas(EstadoActual);
        victoria_diagonais(EstadoActual)).
```

1.3 Defina uma função de utilidade que para um estado terminal que deve retornar o valor do estado.

 1 significa victória por parte do computador
-1 significa victória por parte do jogador
 0 significa empate

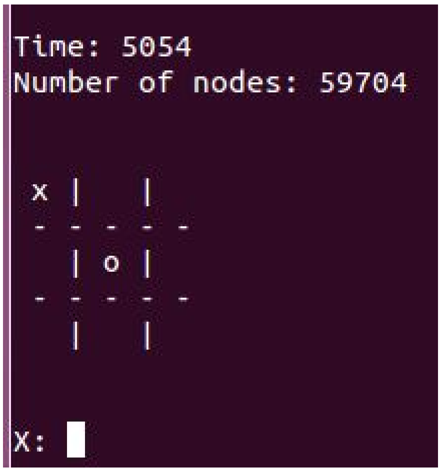
O que esta função de utilidade faz é verificar se alguém ganhou, isto é, se houve uma victória por parte do computador, um derrota ou o jogo acabou em empate.

Nota: O computador é o ‘o’.

```
% ####################################################################################

% valor
valor((EstadoActual, _), 1, P):-
    check_victoria(EstadoActual),
    quem_ganhou(o),!.

valor((EstadoActual, _), -1, P):-
    check_victoria(EstadoActual),
    quem_ganhou(x),!.

valor((EstadoActual, _), 0, P):-
    empate(EstadoActual),!.

% ####################################################################################
```

1.4 Use a implementação da pesquisa minimax dada na aula prática para escolher a melhor jogada num estado.

O minmax escolhe sempre a jogada óptima para aquele estado, portanto ao jogarmos contra o computador, no melhor dos casos resulta sempre num empate. Só por distracção é que pode resultar numa derrota por nossa parte.



Melhor jogada para o estado actual.

1.5 Implemente a pesquisa Alfa-Beta e compare os resultados (tempo e espaço)

Por falta de tempo, não conseguimos implementar o alfa-beta que fosse funcional, ficando este incompleto. Mas por pesquisa percebemos que alfa-beta iria ter um desempenho significamente melhor que minmax em termos de nós visitados.

Na tabela abaixo apresenta-se o número de nós e tempo. Alguns dos estados abaixo, são o mesmo número, pois representa o mesmo número do estado , mas a diferença é que consiste em jogadas em posições diferentes.

Estados	Número de Nós Expandidos	Tempo (ms)
1	59704	4903
1	59704	4798
2	1052	56
2	934	71
2	1052	61
3	33	4
3	39	5
3	46	6
4	2	0
4	4	1

Usando o algoritmo minmax.

1.6 Defina uma função de avaliação que estime o valor de cada estado do jogo.

A função de avaliação que estime o valor do estado do jogo poderá ser, o número de X’s - número de O’s presentes no tabuleiro. Outra função que se ponderou foi o número de sequências (dois X’s) X’s - número de sequências O’s (dois O’s) e se esse não se verificasse, então seria a diferença entre o número de X’s e O’s isolados.

1.7 Implemente um agente inteligente

Consiste em quatro cycles:

- o primeiro cycle, corresponde a uma victoria
- o segundo cycle, correspondente a um empate
- o terceiro cycle, corresponde a um jogada feita pelo computador
- o quarto cycle, corresponde a uma jogada feito pelo Jogador

```
cycle(_,(EstadoActual,Winner)):- (victoria_linhas(EstadoActual);
    victoria_colunas(EstadoActual);
    victoria_diagonais(EstadoActual)),
    prints(EstadoActual),
    format('Vencedor: ~w\n',[Winner]),!.

cycle(_,(EstadoActual,_)):-
    empate(EstadoActual),
    prints(EstadoActual),
    write('Empate! lel'),nl,!.

cycle(0,(EstadoActual,V)):-

    prints(EstadoActual),
    statistics(real_time,[Ti,_]),
    minimax_decidir((EstadoActual,V),Op),
    statistics(real_time,[Tf,_]), T is Tf-Ti,

    nodes(Nodes),
    format('\nTime: ~w\n',[T]),
    format('Number of nodes: ~w\n\n',[Nodes]),
    resetNodes,
    opl((EstadoActual,V),Op,EstadoActualSeguinte),
    cycle(1,EstadoActualSeguinte).

cycle(1,(EstadoActual,V)):-
    prints(EstadoActual),
    write('\nX: '),
    read(X),
    write('Y: '),
    read(Y),
    chooseOp(X,Y,Op),
    opl((EstadoActual,V),Op,EstadoActualSeguinte),
    cycle(0,EstadoActualSeguinte).

chooseOp(X,Y,Op):-
    (X = 1, Y = 1 -> Op = jogar11;
        X = 1, Y = 2 -> Op = jogar12;
        X = 1, Y = 3 -> Op = jogar13;
        X = 2, Y = 1 -> Op = jogar21;
        X = 2, Y = 2 -> Op = jogar22;
        X = 2, Y = 3 -> Op = jogar23;
        X = 3, Y = 1 -> Op = jogar31;
        X = 3, Y = 2 -> Op = jogar32;
        X = 3, Y = 3 -> Op = jogar33).
```

O agente inteligente está sempre em loop, até houver uma derrota/victoria ou empate. No ciclo do jogador é pedido um input das coordenadas. Com essas coordenadas, é escolhido o operador (através de ChooseOp/3) e é feito a jogada. Em seguinte verifica-se se houve uma victória (o primeiro cycle), ou um empate (o segundo cycle), caso contrário é a vez do computador. O computador consegue realizar um jogada usando minimax_decidir/2 para, usando o estado actual, escolher a melhor jogada (melhor Operador). Após executar a jogada, repete novamente o ciclo descrito anteriormente, só que passa a jogada para o Jogador.

1.8 Apresente uma tabela com o número de nós expandidos para diferentes estados

Na tabela abaixo apresenta-se o número de nós e tempo. Alguns dos estados abaixo, são o mesmo número, pois representa o mesmo número do estado , mas a diferença é que consiste em jogadas em posições diferentes.

Estados	Número de Nós Expandidos	Tempo (ms)
1	59704	4903
1	59704	4798
2	1052	56
2	934	71
2	1052	61
3	33	4
3	39	5
3	46	6
4	2	0
4	4	1

Usando o algoritmo minmax

2. Jogo do 4 em linha

1.1. Escolha uma estrutura de dados para representar os estados.

Tal como no jogo do galo, escolheu-se uma estrutura de dados que passa por ser um tuplo de todos elementos. O primeiro elemento é uma lista que contém as posições do tabuleiro 4x4, e o segundo elemento contém a informação de quem jogou.

```
estado_inicial((
[(p(1,1), b),(p(2,1), b),(p(3,1), b),(p(4,1),b),
 (p(1,2), b),(p(2,2), x),(p(3,2), o),(p(4,2),b),
 (p(1,3), b),(p(2,3), x),(p(3,3), o),(p(4,3),b),
 (p(1,4), o),(p(2,4), x),(p(3,4), o),(p(4,4),b)], o)).
```

Estado inicial.

1.2 Estado terminal

O jogo pode terminar quando algum jogador ganha por linha ou coluna. Pode também acabar em caso de empate. O código referente a esse decisão é o seguinte:

```
%verifys if game ends in lines row or diagonals, or if it draws.
terminal((B,_)):-
    draw(B);rows(B);lines(B).

lines(B):-
    (findall(X1,(member(X1,B),X1 = (p(1,_),_)),L1)),verify(L1, 0, 0,4);
    (findall(X2,(member(X2,B),X2 = (p(2,_),_)),L2)),verify(L2, 0, 0,4);
    (findall(X3,(member(X3,B),X3 = (p(3,_),_)),L3)),verify(L3, 0, 0,4);
    (findall(X4,(member(X4,B),X4 = (p(4,_),_)),L4)),verify(L4, 0, 0,4).

rows(B):-
    (findall(X1,(member(X1,B),X1 = (p(_,1),_)),L1)),verify(L1, 0, 0,4);
    (findall(X2,(member(X2,B),X2 = (p(_,2),_)),L2)),verify(L2, 0, 0,4);
    (findall(X3,(member(X3,B),X3 = (p(_,3),_)),L3)),verify(L3, 0, 0,4);
    (findall(X4,(member(X4,B),X4 = (p(_,4),_)),L4)),verify(L4, 0, 0,4).
```

```
%verifys number of x's or o's in a line, diagonal or row. If both below 4, it fails.
%List of cases:
%End of line 0 houses left, 4x, win.
%End of line,0 houses left; 4o, win.
%in line. house is X. increment x
%in line, house is o, increment o
%in line, house is empty, increment none
verify( [], 4, 0,0):-makeWin(winner(x)),!.
verify( [], 0, 4,0):-makeWin(winner(o)),!.

verify([(p(_,_),x)|T], NumberOfX, NumberOfO,HousesLeft):-
        NumberOfXTemp is NumberOfX + 1,
        HousesLeftTemp is HousesLeft -1,
        verify(T,NumberOfXTemp,NumberOfO,HousesLeftTemp),!.

verify([(p(_,_),o)|T], NumberOfX, NumberOfO,HousesLeft):-
        NumberOfOTemp is NumberOfO + 1,
        HousesLeftTemp is HousesLeft -1,
        verify(T,NumberOfX,NumberOfOTemp,HousesLeftTemp),!.

verify([(p(_,_),b)|T], NumberOfX, NumberOfO,HousesLeft):-
        HousesLeftTemp is HousesLeft -1,
        verify(T,NumberOfX,NumberOfO,HousesLeftTemp),!.
```

1.3 Defina uma função de utilidade que para um estado terminal que deve retornar o valor do estado.

Segue a mesma lógica que o jogo do galo.

1 representa vitória do computador
-1 representa vitória do jogador
0 representa empate

```
valor((E, _), 1, _):-(lines(E);rows(E)),winner(o),!.
valor((E, _), -1, _):-(lines(E);rows(E)),winner(x),!.
valor((E, _), 0, _):- draw(E),!.
```

1.4 Use a implementação da pesquisa minimax dada na aula prática para escolher a melhor jogada num estado.

O algoritmo minmax garante a melhor jogada dado um estado. Como tal no melhor caso só se consegue empatar, e por distracção, perder, por parte do Jogador.

x - Jogador
o - Computador
b - Espaço em branco

```
b | b | b | b |
b | b | o | o |
x | x | o | x |
o | x | o | b |

X: 3.
b | b | x | b |
b | b | o | o |
x | x | o | x |
o | x | o | b |

Time: 502
Number of nodes: 2292

b | b | x | b |
b | o | o | o |
x | x | o | x |
o | x | o | b |
```

exemplo da melhor jogada no estado actual.

1.5 Implemente a pesquisa Alfa-Beta e compare os resultados (tempo e espaço)

Como referido anteriormente no jogo do galo, Por falta de tempo, não conseguimos implementar o alfa-beta que fosse funcional, ficando este incompleto.

Estados	Número de Nós Expandidos	Tempo (ms)
1	301404	86508
1	264324	56912
2	672	68
2	672	65
2	672	61
3	8	0
3	8	1
3	8	1
4	2	0
4	2	0

1.6 Defina uma função de avaliação que estime o valor de cada estado do jogo.

Tal com no jogo do galo, ponderamos que uma função aceitável fosse uma função que estime o valor do estado do jogo poderá ser, o número de X’s - número de O’s presentes em cada linha e coluna no tabuleiro. Outra função que se ponderou foi o número de sequências (dois X’s) X’s - numero de sequencias O’s (dois O’s) e se esse não se verificasse, então seria a diferença entre o número de X’s e O’s isolados.

1.7 Implemente um agente inteligente

Como referido no jogo do galo, o agente inteligente consiste de 4 cycles.

```
cycle(_,(EstadoActual,Winner)):- (lines(EstadoActual);
    rows(EstadoActual)),
    tprint(EstadoActual),
    format('Vencedor: ~w\n',[Winner]),!.

cycle(_,(EstadoActual,_)):-
    draw(EstadoActual),
    tprint(EstadoActual),
    write('Empate! lel'),nl,!.

cycle(0,(EstadoActual,V)):-

    tprint(EstadoActual),
    statistics(real_time,[Ti,_]),
    minimax_decidir((EstadoActual,V),Op),
    statistics(real_time,[Tf,_]), T is Tf-Ti,

    nodes(Nodes),
    format('\nTime: ~w\n',[T]),
    format('Number of nodes: ~w\n\n',[Nodes]),
    resetNodes,
    op1((EstadoActual,V),Op,EstadoActualSeguinte),
    cycle(1,EstadoActualSeguinte).

cycle(1,(EstadoActual,V)):-
    write('\n'),
    tprint(EstadoActual),
    write('\nX: '),
    read(X),
    write('\n'),
    chooseOp(X,Op),
    op1((EstadoActual,V),Op,EstadoActualSeguinte),
    cycle(0,EstadoActualSeguinte).

tprint([]).
tprint([(p(X,Y),V)|T]) :-
        write(V), write(' | '),
        X=4-> nl, tprint(T); tprint(T).
```

```
op1((EstadoActual,O),jogar1, (EstadoSeguinte,P)):-
    (O = x -> P = o; P = x),
    insere(1, P, EstadoActual,EstadoSeguinte).

op1((EstadoActual,O),jogar2, (EstadoSeguinte,P)):-
    (O = x -> P = o; P = x),
    insere(2, P, EstadoActual,EstadoSeguinte).

op1((EstadoActual,O),jogar3, (EstadoSeguinte,P)):-
    (O = x -> P = o; P = x),
    insere(3, P, EstadoActual,EstadoSeguinte).

op1((EstadoActual,O),jogar4, (EstadoSeguinte,P)):-
    (O = x -> P = o; P = x),
    insere(4, P, EstadoActual,EstadoSeguinte).

chooseOp(X,Op):-
        (X = 1 -> Op = jogar1;
         X = 2 -> Op = jogar2;
         X = 3 -> Op = jogar3;
         X = 4 -> Op = jogar4).
```

O funcionamento é estritamente igual, apenas variando no numero de operadores.

1.8 Apresente uma tabela com o número de nós expandidos para diferentes estados

Na tabela abaixo apresenta-se o número de nós e tempo. Alguns dos estados abaixo, são o mesmo número, mas a diferença é que consiste em jogadas em posições diferentes.

Estados	Número de Nós Expandidos	Tempo (ms)
1	301404	86508
1	264324	56912
2	672	68
2	672	65
2	672	61
3	8	0
3	8	1
3	8	1
4	2	0
4	2	0

Conclusão

Depois de termos explorado estes conceitos base de teoria da decisão, e juntado a isso o número de experiências e debate interno que tivemos como um grupo sobre como abordar melhor o tópico, podemos afirmar com certeza que obtemos as metas de aprendizagem pessoal que nos foram apresentadas, tal como obtemos uma perspectiva diferente sobre a área em questão e como abordar problemas do gênero.

Afirmamos, sem duvida, que o conhecimento que obtemos neste trabalho nos vai acompanhar para outras áreas da nossa carreira, seja esta académica ou de trabalho.