3º Trabalho de Inteligência Artificial

2018/2019

Jogos de dois jogadores – Jogos com informação completa determinísticos

Docente:
Irene Pimenta Rodrigues

Realizado por:

Daniel Serrano – 35087

Miguel Serrano - 34149

09 de abril de 2019

Conteúdo

Introdução:

Neste trabalho, no âmbito da cadeira de Inteligência Artificial iremos resolver o jogo do nim, jogo proposto pela professora, e o jogo do galo, jogo escolhido por nós visto termos começado a implementá-lo numa aula prática. Para resolução destes jogos por parte do CPU iremos utilizar os algoritmos de minmax e alfabeta.

1º Exercício:

1.a)

A estrutura de dados escolhida por nós, para representar os estados do jogo foi a seguinte:

```
estado_inicial(0,[]):-!.
estado_inicial(X,S0):-
  X2 is X*2-1,
  X1 is X-1,
  append(S01,[X2],S0),
  estado_inicial(X1,S01).
```

Estrutura esta que recebe como input um valor X que equivale ao número de colunas e a partir desse valor coluna 2*X -1 peças em jogo por cada coluna, e diminui 1 valor esse mesmo X, de modo a que a última coluna a colocar peças fique apenas com uma peça. Exemplo: Se X=3 o estado será do tipo: [1,3,5].

1.b)

Para determinar o estado terminal utilizámos o predicado:

```
terminal(A,Res):-
  soma(A,Res),
  Res = 0.

soma([],0).
soma([A|R],Res):-
  soma(R,Res1),
  Res is (A+Res1).
```

Em que é calculada a soma das peças em todas as colunas e o estado será então terminal, e de acordo com as regras do jogo, quando o número de peças for 0.

1.c)

Para definir a função utilidade utilizámos o seguinte predicado:

```
valor(0, 1, P):-
  X is P mod 2,
  X = 0,
  !.

valor(0, -1, _):-!.
```

Função que ao atingir um estado terminal verifica se a profundidade é par ou não. Caso seja par, a função retorna o valor 1 o que significa que o jogador principal ganhou. Caso seja ímpar, retorna o valor -1 e o jogador vitorioso é o adversário. De reparar que não temos um caso de empate pois isso é impossível neste jogo.

1.d)

Para este ponto foi necessário então implementar o algoritmo de minimax. Para tal, utilizámos o algoritmo disponibilizado pela professora, mas realizámos algumas alterações de modo a que se adaptasse ao nosso jogo.

Para utilizar este algoritmo para escolher a melhor jogada teremos de seguir os seguintes passos:

1. ?- [main].
2. ?- main.
3. Introduza o número de linhas: 3. (à escolhe do user)
4. Escolha Algoritmo: \n 1. Minimax \n 2. AlfaBeta \n : 1.
5. Escolha Modo: \n 1. Melhor Jogada \n 2. Homem vs CPU \n 3. CPU vs CPU \n : 1.
6. Insira a Profundidade: 10. (à escolha do user)
7. Output do programa:

```
Estado Inicial:
coluna 1: []
coluna 2: [][][]
coluna 3: [][][][][]


Melhor Jogada:
retiraColuna(3-3)

coluna 1: []
coluna 2: [][][]
coluna 3: [][]
```

1.e)

1.f)

Para a função de avaliação consultámos um site (link) em que conseguimos perceber o que a esta função devia testar:

> **In fact, we can predict the winner of the game before even playing the game !**
>
> **Nim-Sum :** The cumulative XOR value of the number of coins/stones in each piles/heaps at any point of the game is called Nim-Sum at that point.
>
> ***"If both A and B play optimally (i.e- they don't make any mistakes), then the player starting first is guaranteed to win if the Nim-Sum at the beginning of the game is non-zero. Otherwise, if the Nim-Sum evaluates to zero, then player A will lose definitely."***
>
> For the proof of the above theorem, see-
> https://en.wikipedia.org/wiki/Nim#Proof_of_the_winning_formula
>
> Let us apply the above theorem in the games played above. In the first game **A** started first and the Nim-Sum at the beginning of the game was, 3 XOR 4 XOR 5 = 2, which is a non-zero value, and hence **A** won. Whereas in the second game-play, when the initial configuration of the piles were 1, 4, and 5 and **A** started first, then **A** was destined to lose as the Nim-Sum at the beginning of the game wasv1 XOR 4 XOR 5 = 0 .

Portanto, o que esta função irá fazer é fazer a operação xor entre o número de elementos de cada coluna, e, caso o resultado seja 0 significa que estamos num estado de vantagem, ou seja, Se estivermos numa profundidade par, e o valor da função nim-sum for 0 significa que o jogador principal está em vantagem, no entanto também estará em vantagem se numa profundidade impar o valor da função nim-sum não der 0. Se nestes casos em que quem está em vantagem é o jogador principal a função avaliação retornará o valor 1, caso contrário retornará o valor -1.

```
f_avalia(Estado,Val,P):- nim_sum(Estado,Val1),bom_mau2(Val1,Val,P).

 nim_sum([V],V).   %% calcula o xor entre todos os elementos
 nim_sum([A,B|R],V):-
   C is A^B,
   nim_sum([C|R],V),!.

 bom_mau2(0,1,P):-
   Y is P mod 2,
   Y =0,!.
 bom_mau2(0,-1,P):-
   Y is P mod 2,
   Y \=0,!.
 bom_mau2(X,0,_):-
   X\=0,!.
```

1.g)

Para a implementação de um agente inteligente optámos por permitir que o utilizador pudesse jogar contra um PC ou então visualizar o jogo entro 2 PC's. Este agente pode ser mais difícil se a profundidade limite definida pelo utilizador no início de cada jogo for maior.
De referir que num jogo entre dois PC's irá ganhar aquele que primeiro atingir um estado de vantagem.

```
jogadorP(_,S0):-
    terminal(S0,_),write('Jogador Adversário venceu').

jogadorP(2,S0):-
    write('Introduza a coluna de onde pretende remover: '),
    read(Coluna),nl,
    write('Introduza o numero de pecas que pretende remover: '),
    read(Pecas),nl,
    Op = retiraColuna(Coluna-Pecas),
    op1(S0,Op,S1),
    write('Jogada do Jogador Principal: '),nl,
    write(Op),nl,nl,
    desenha_estado(S1,1),nl,nl,
    jogadorA(2,S1).


jogadorP(3,S0):-
    decidir_jogada(S0, Op),!,
    op1(S0,Op,S1),!,
    write('Jogada do Jogador Principal: '),nl,
    write(Op),nl,nl,
    desenha_estado(S1,1),!,nl,nl,
    jogadorA(3,S1).


jogadorA(_,S0):-
    terminal(S0,_),write('Jogador Principal venceu').

jogadorA(Modo,S0):-
    decidir_jogada(S0, Op),!,
    op1(S0,Op,S1),!,
    write('Jogada do Jogador Adversário: '),nl,
    write(Op),nl,nl,
    desenha_estado(S1,1),nl,
    jogadorP(Modo,S1).
```

1.h)

2º Exercício:

Para o 2º exercício era pedido um jogo de dois jogadores, e como já tínhamos iniciado a programação do jogo do galo num das aulas práticas decidimos continuá-lo neste pronto.

2.a)

A estrutura de dados escolhida por nós, para representar os estados do jogo foi a seguinte:

```
estado_inicial(([(p1,_), (p2,_), (p3,_),
                 (p4,_), (p5,_), (p6,_),
                 (p7,_), (p8,_), (p9,_)],_)).
```

Estrutura esta que como todos os jogos do galo, o estado inicial não tem nenhuma posição preenchida e as posições são representadas por pPos.

2.b)

Para determinar o estado terminal utilizámos o predicado:

```
terminal((E,_)):-linhas(E); colunas(E);
                 diagonais(E);empate(E).
```

Em que verifica se há alguma sequencia de símbolos iguais tanto nas linhas, como nas colunas, como nas diagonais, e caso isso não se verifique, irá também verificar se ocorreu algum empate. Este predicado é auxiliado pelas seguintes funções:

```
linhas(E):-
    (tres_seguidos(E,p1,p2,p3);
    tres_seguidos(E,p4,p5,p6);
    tres_seguidos(E,p7,p8,p9)).

colunas(E):-
    (tres_seguidos(E,p1,p4,p7);
    tres_seguidos(E,p2,p5,p8);
    tres_seguidos(E,p3,p6,p9)).

diagonais(E):-
    (tres_seguidos(E,p1,p5,p9);
    tres_seguidos(E,p3,p5,p7)).


empate(E):-
    com_valor(E), asserta(empate).
```

```
tres_seguidos(E,X,Y,Z):-member((X,P1),E),
                 member((Y,P2),E),
                 member((Z,P3),E),
                 atom(P1), atom(P2), atom(P3),
                 L = [P1, P2, P3],
                 compare(L),
                 save(winner(P1)),!.

compare([X|Y]):-aux(Y,X).
compare([]).

aux([],_).
aux([Y|X],Y) :- aux(X,Y).


com_valor([]).
com_valor([(_,K)|T]):-
    atom(K),
    com_valor(T).
```

2.c)

Para definir a função utilidade utilizámos o seguinte predicado:

```
valor((E,_),1,_):- (linhas(E); colunas(E); diagonais(E)), winner(o),!.
valor((E,_),-1,_):-(linhas(E); colunas(E); diagonais(E)), winner(x),!.
valor((E,_),0,_):- empate(E),!.
```

Função verifica a profundidade na arvora de pesquisa e os casos em que o estado é terminal, com a exceção de empate. Os valores devolvidos pela mesma podem ser 1, 0 ou -1 sendo que 0 representa empate, 1 ganha e -1 perde.

2.d)

Para este jogo apenas optámos por não colocar nenhum método de visualizar 2 PC's a jogar um contra o outro, mas apenas um utilizador contra um PC. A função que nos permite isto é a seguinte:

```
play(_,(E,J)):- (linhas(E);colunas(E);diagonais(E)), print_(E),write('Vencedor: '),write(J),!.
play(_,(E,_)):- empate(E), print_(E),write('Empate!'),nl,!.

play('p',(E,J)):-
    print_(E),
    nl,statistics(real_time,[Ti,_]),
    minimax_decidir((E,J),Op),
    statistics(real_time,[Tf,_]), T is Tf-Ti,nl,
    write('Tempo: '(T)),nl,
    n(N),
    write('Numero de Nos: '(N)),initInc, nl,
    write(Op),nl,
    op1((E,J),Op,Es),
    play('h',Es).

play('h',(E,J)):-
    print_(E),nl,
    write('Escreva a posicao onde deseja jogar: '),
    read(X),
    trocaSimbolo(J,J1),
    (X=1    ->op1((E,J),insere(p1,J1),Es),!
    ;(X=2   ->op1((E,J),insere(p2,J1),Es),!
    ;(X=3   ->op1((E,J),insere(p3,J1),Es),!
    ;(X=4   ->op1((E,J),insere(p4,J1),Es),!
    ;(X=5   ->op1((E,J),insere(p5,J1),Es),!
    ;(X=6   ->op1((E,J),insere(p6,J1),Es),!
    ;(X=7   ->op1((E,J),insere(p7,J1),Es),!
    ;(X=8   ->op1((E,J),insere(p8,J1),Es),!
    ;(X=9   ->op1((E,J),insere(p9,J1),Es),!
    ;(write('valor invalido!'),nl, play('h',(E,J))))))))),
    play('p',Es).
```

2.e)

MINIMAX	ESTADO
59704	Posição inicial: 1
63904	Posição inicial: 2
59704	Posição inicial: 3
63904	Posição inicial: 4
55504	Posição inicial: 5
63904	Posição inicial: 6
59704	Posição inicial: 7
63904	Posição inicial: 8
59704	Posição inicial: 9
934	Posição inicial: 1 + 2º posição: 1
46	Posição inicial: 1 + 2º posição: 2 + 3º Posição: 7
4	Posição inicial: 1 + 2º posição: 2 + 3º Posição: 7 + 4ª Posição: 6

Conclusão:

Com a realização deste trabalho ficámos a ter mais conhecimento sobre os tipos de jogos com dois jogadores, e apesar de não termos conseguido implementar o algoritmo de alfabeta conseguimos calcular que seria muito mais eficiente que o minimax, visto este ter sido bastante lento quando se tratar de jogos muito exigentes, como se verificou no jogo do galo.