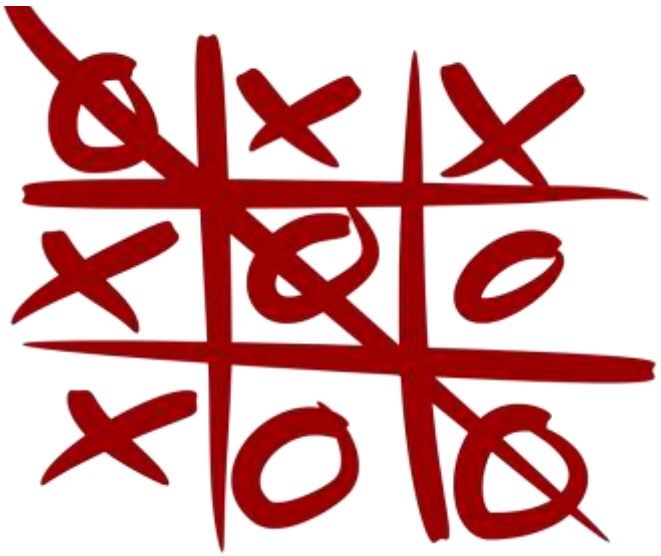
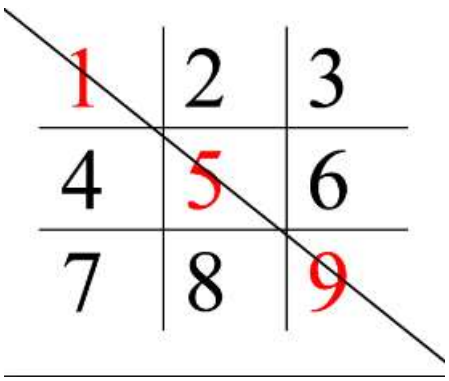
Inteligência Artificial

Trabalho 2015/2016

09/05/16

UNIVERSIDADE DE ÉVORA





Professora:
Irene Pimenta Rodrigues

Realizado por:
João Calhau - 31621
José Pimenta - 31677

Índice

Introdução

Este trabalho enquadra-se na disciplina de Inteligência Artificial e vamos abordar minimax e alfa-beta para resolução de um exercicio definido pela professora, em que temos de resolver o jogo do galo e outro jogo escolhido por nós.
Iremos assim tentar dar o nosso melhor, e iremos tentar utilizar os métodos que achemos mais correctos ou propícios à boa evolução do trabalho e que no final se concretize o que nos é pedido.

1. Jogo do Galo

1.1. Questões

1.1.1. Escolha uma estrutura de dados para representar os estados dos dois jogos.

A estrutura utilizada para representar o estado do jogo do galo é um tuplo com uma lista de posições tabuleiro onde cada posição contem uma referencia (um valor não instanciado), e um caracter “x” ou “o” que indica uma jogada e a ultima peça que foi jogada.

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

1.1.2. Defina o predicado terminal (estado) que sucede quando o estado é termina para cada jogo.

Um estado é terminal se há uma linha, coluna ou diagonal completa, seja ela com x’s ou com o’s, ou no ultimo dos casos quando todas as posições estão preenchidas, ou seja, quando há um empate. Para o segundo jogo ainda não está definido.

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

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

A função de utilidade 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.

```
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),!.
```

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

Com o algoritmo minimax as jogadas são sempre óptimas, logo se jogarmos contra o computador, no melhor dos casos conseguimos um empate.

1.1.5. Implemente a pesquisa Alfa-Beta e compare os resultados (tempo e espaço) em exemplos com os dois jogos.

Através dos resultados observados, podemos concluir que o minimax, apesar de demorar mais tempo a efetuar as jogadas, faz sempre uma jogada ótima, enquanto que o corte alfa-beta não faz a melhor jogada ótima, mas leva menos tempo a efetuar a mesma (Isto talvez devido a uma má implementação do mesmo).

Posições preenchidas	Tempo(ms)	Nº de nós
1	1984	55504
2	27	11722
3	2	50

Tabela 1: Resultados para o algoritmo minimax

Posições preenchidas	Tempo(ms)	Nº de nós
1	6	196
2	1	36
3	1	16

Tabela2: Resultados para o algoritmo alfa-beta

Tal como se pode observar é notável a redução e tempo e nº de nós necessários para efetuar uma jogada. Apesar destes resultados não conseguimos colocar o alfabeta a fazer sempre a melhor jogada. Podemos então dizer que temos um jogar esperto e um jogador burro, o minimax e o alfa-beta, respetivamente.

1.1.6. Defina uma função de avaliação que estime o valor de cada estado do jogo, use os dois algoritmos anteriores com corte em profundidade e compare os resultados (tempo e espaço), com exemplos dos dois jogos.

A função avaliação feita para este jogo foi fazer o calculo de peças em jogo, por linhas, colunas e diagonais (consoante o numero de peças maior o peso adicionado). Fazer a mesma operação para o oponente e subtrair um valor ao outro.

```
func_aval((E,J), Val,_):-
    trocaSimbolo(J, J2),
    aval(E,J2,Val).

aval(E, J, Val):-
    linhas2(E,J, Val1),
    trocaSimbolo(J, J2),
    linhas2(E,J2,Val2),
    Val is (Val1-Val2).
```

1.1.7. Implemente um agente inteligente que joga os dois jogos, usando a pesquisa definida na alínea anterior.

Implementamos um agente inteligente que joga contra o jogador num ciclo até que se verifique o terminal.

```
jogada_minimax(_,(E,J)):- (linhas(E);colunas(E);diagonais(E)), print_(E),write('WINNER: '),write(J),!.
jogada_minimax(_,(E,_)):- empate(E), print_(E),write('TIE!'),nl,!.

jogada_minimax('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,
    nl,
    op1((E,J),Op,Es),
    jogada_minimax('h',Es).

jogada_minimax('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, jogada_minimax('h',(E,J)))))))))),
    jogada_minimax('p',Es).
```

2. Jogo do galo com números

2.1. Questões

2.1.1. Escolha uma estrutura de dados para representar os estados dos dois jogos.

A estrutura utilizada para representar o estado do jogo do galo é um tuplo com uma lista de posições tabuleiro onde cada posição contem uma referencia (um valor não instanciado), uma lista com os caracteres já utilizados e um caracter com o ultimo numero que foi jogado.

```
estado_inicial(([(p(1),_), (p(2),_), (p(3),_),
                 (p(4),_), (p(5),_), (p(6),_),
                 (p(7),_), (p(8),_), (p(9),_)],[0],_)).
```

2.1.2. Defina o predicado terminal (estado) que sucede quando o estado é termina para cada jogo

Um estado é terminal se há uma linha, coluna ou diagonal completa, seja ela com números que somados deem 15, ou no ultimo dos casos quando todas as posições estão preenchidas, ou seja, quando há um empate.

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

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

A função de utilidade 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.

```
%função de utilidade, retorna o valor dos estados terminais, 1 ganha -1 perde
valor((E,_,_),1,_):- (linhas(E); colunas(E); diagonais(E)), winner(impar),!.
valor((E,_,_),-1,_):-(linhas(E); colunas(E); diagonais(E)), winner(par),!.
valor((E,_,_),0,_):- empate(E),!.
```

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

Neste jogo o minimax não é a escolha mais acertada para jogar, devido ao grande numero de estados possíveis, o jogo acaba por crashar antes de poder jogar qualquer coisa. A única maneira que conseguimos por o jogo a jogar foi dando valores iniciais ás várias posições (como se o jogo já fosse a meio).

2.1.5. Implemente a pesquisa Alfa-Beta e compare os resultados (tempo e espaço) em exemplos com os dois jogos.

Devido ao alfa-beta estar mal implementado, ele consegue jogar, mas não faz sempre a melhor jogada possível. Resultados com o algoritmo minimax não disponíveis devido ao programa parar e não fazer jogada nenhuma.

Posições preenchidas	Tempo(ms)	Nº de nós
3	12	604
4	1	56
5	1	4

Tabela 1: Resultados com o algoritmo alfa-beta

Podemos notar uma quantia no numero de nós muito mais elevada do que no jogo do galo normal, devido á complexidade deste jogo.

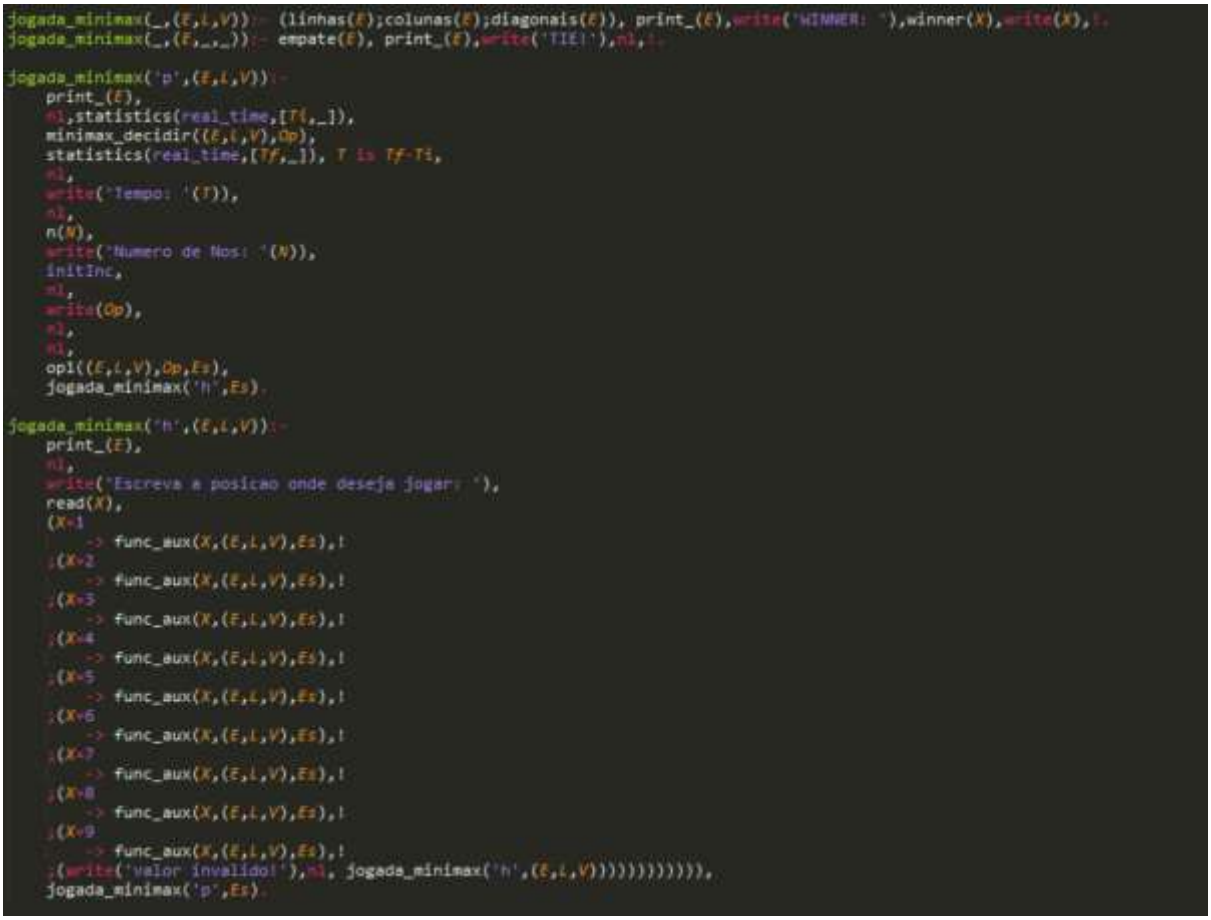
2.1.6. Defina uma função de avaliação que estime o valor de cada estado do jogo, use os dois algoritmos anteriores com corte em profundidade e compare os resultados (tempo e espaço), com exemplos dos dois jogos.

A função avaliação feita para este jogo foi fazer o calculo do numero de peças impares e pares em jogo e subtrair uma á outra, visto que o jogador é sempre as peças impares e joga primeiro, haverá sempre um numero de peças impares maior que pares.

```
func_aval((E,L,V), Val,_):-
    countImpares(L,X),
    countPares(L,Y),
    Val is Y-X.
```

2.1.7. Implemente um agente inteligente que joga os dois jogos, usando a pesquisa definida na alínea anterior.

Implementamos um agente inteligente que joga contra o jogador num ciclo até que se verifique o terminal.



```
jogada_minimax(_,(E,L,V)):- (linhas(E);colunas(E);diagonais(E)), print_(E),write('WINNER: '),winner(X),write(X),!.
jogada_minimax(_,(E,_,_)):- empate(E), print_(E),write('TIE!'),nl,!.

jogada_minimax('p',(E,L,V)):-
    print_(E),
    nl,statistics(real_time,[Ti,_]),
    minimax_decidir((E,L,V),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,
    nl,
    opl((E,L,V),Op,Es),
    jogada_minimax('h',Es).

jogada_minimax('h',(E,L,V)):-
    print_(E),
    nl,
    write('Escreva a posicao onde deseja jogar: '),
    read(X),
    (X=1
        -> func_aux(X,(E,L,V),Es),!
    ;(X=2
        -> func_aux(X,(E,L,V),Es),!
    ;(X=3
        -> func_aux(X,(E,L,V),Es),!
    ;(X=4
        -> func_aux(X,(E,L,V),Es),!
    ;(X=5
        -> func_aux(X,(E,L,V),Es),!
    ;(X=6
        -> func_aux(X,(E,L,V),Es),!
    ;(X=7
        -> func_aux(X,(E,L,V),Es),!
    ;(X=8
        -> func_aux(X,(E,L,V),Es),!
    ;(X=9
        -> func_aux(X,(E,L,V),Es),!
    ;(write('valor invalido!'),nl, jogada_minimax('h',(E,L,V)))))))))),
    jogada_minimax('p',Es).
```

3. Apresente uma tabela com o número de nós expandidos para diferentes estados dos 2 jogos com os vários algoritmos

Minimax (jogo do galo)	Alfa-beta (jogo do galo)	Minimax (jogo do galo com números	Alfa-beta (jogo do galo com números)	Jogada inicial na posição X
59704	252	-	604	1
63904	223	-	604	2
59704	223	-	604	3
63904	174	-	604	4
55504	196	-	-	5
63904	179	-	604	6
59704	263	-	604	7
63904	215	-	604	8
59704	230	-	-	9

Tabela 1: Números de nós para os diferentes jogos e algoritmos

Conclusão

Após a realização deste trabalho, ficámos a conhecer melhor como funcionam algoritmos de minimax e alfa-beta. Aplicámos conhecimentos adquiridos de modo a tentar resolver os problemas propostos pela professora, e conseguimos resolver minimamente os problemas propostos. Conseguimos resolver o Jogo do Galo em minimax e alfa-beta (apesar de ter alguns erros) em bom tempo, e conseguimos resolver o Jogo do Galo com números em minimax (com variáveis já instanciadas, isto porque é um jogo muito exigente para o minimax) e alfa-beta.
Tivémos assim dificuldades em implementar o algoritmo alfa-beta que funcionasse 100%, e o minimax não foi o algoritmo adequado para tipos de jogos muito exigentes, como se verificou no Jogo do Galo com números em que temos de inicializar algumas variáveis de modo ao jogo ser possível de ser jogado.