UNIVERSIDADE DE ÉVORA

Inteligência Artificial

Trabalho 3

Miguel Grilo 58387

Jorge Couto 58656

Colégio Luís António Verney

Tabela de Conteúdos

Exercício 1

Alínea a)

- $\%estado\_inicial(e(Tabuleiro, Jogador)).$
  $estado\_inicial(e([v, v, v, v, v, v, v, v, v], x)).$
- $inv(x, y).$
  $inv(y, x).$

Alínea b)

- $\%Estados\_terminais$
  $terminal(e(L, \_)) :- linha(e(L, \_)).$
  $terminal(e(L, \_)) :- coluna(e(L, \_)).$
  $terminal(e(L, \_)) :- diagonal(e(L, \_)).$

- $\%Linhas$
  $linha(e([s, o, s, \_, \_, \_, \_, \_, \_], \_)).$
  $linha(e([\_, \_, \_, s, o, s, \_, \_, \_], \_)).$
  $linha(e([\_, \_, \_, \_, \_, \_, s, o, s], \_)).$
  $\%Colunas$
  $coluna(e([s, \_, \_, o, \_, \_, s, \_, \_], \_)).$
  $coluna(e([\_, s, \_, \_, o, \_, \_, s, \_], \_)).$
  $coluna(e([\_, \_, s, \_, \_, o, \_, \_, s], \_)).$
  $\%Diagonal$
  $diagonal(e([s, \_, \_, \_, o, \_, \_, \_, s], \_)).$
  $diagonal(e([\_, \_, s, \_, o, \_, s, \_, \_], \_)).$

Alínea c)

- $\%Funcao\_utilidade$
  $valor(e(Tabuleiro, \_), 0, \_) :- \%Empate$
  $\quad \backslash + \ terminal(e(Tabuleiro, \_)),$
  $\quad \backslash + \ member(v, Tabuleiro).$
  $valor(e(Tabuleiro, \_), V, P) :-$
  $\quad terminal(e(Tabuleiro, \_)),$
  $\quad X \ is \ P \ mod \ 2,$
  $\quad (X == 1, V = 1; \ \%Ganha$
  $\quad X == 0, V = -1). \%Perde$

Alínea d)

Consultamos os ficheiros $[sos]$. e $[minmax]$. para simular um jogo a partir de um dado $estado\_inicial$ e escolhendo a melhor jogada em cada caso com recurso ao algoritmo $minmax$.

- $estado\_inicial(e([s, v, v, v, v, s, v, v, o], x)).$

S		
		S
		O

$|\ ?- g('sos.pl').$

$joga(8, o)$

- $estado\_inicial(e([s, v, v, v, v, s, v, o, o], y)).$

S		
		S
	O	O

$|\ ?- g('sos.pl').$

$joga(7, o)$

- $estado\_inicial(e([s, v, v, v, v, s, o, o, o], x)).$

S		
		S
O	O	O

$|\ ?- g('sos.pl').$

$joga(5, o)$

- $estado\_inicial(e([s, v, v, v, o, s, o, o, o], y)).$

S		
	O	S
O	O	O

$| ? - g('sos.pl').$

$joga(4, o)$

- $estado\_inicial(e([s, v, v, o, o, s, o, o, o], x)).$

S		
O	O	S
O	O	O

$| ? - g('sos.pl').$

$joga(2, o)$

- $estado\_inicial(e([s, o, v, o, o, s, o, o, o], y)).$

S	O	
O	O	S
O	O	O

$| ? - g('sos.pl').$

$joga(3, s)$

- $estado\_final(e([s, o, s, o, o, s, o, o, o], x)).$

S	O	S
O	O	S
O	O	O

$Linha\ 1\ forma\ S\ O\ S$

$Vitória\ do\ Jogador\ y$ (último a jogar)

Alínea e)

- ```
  :- dynamic(nos_visitados/1).
  inicializa_contador :-
          retractall(nos_visitados(_)),
          asserta(nos_visitados(0)).

  inc :-
          retract(nos_visitados(N)),
          N1 is N + 1,
          asserta(nos_visitados(N1)).

  g(Jogo):- [Jogo], estado_inicial(Ei), alfabeta(Ei, Op), write(Op), nl.

  alfabeta(Ei, terminou) :- terminal(Ei).
  alfabeta(Ei, MelhorJogada) :- %x_MAX
          findall(V - Op, (op1(Ei, Op, Es),
                  alfabeta_min(Es, V, 1, -10000, 10000)), L),
          escolhe_max(L, MelhorJogada).

  alfabeta_min(Ei, Val, P, _, _) :- %terminal_MIN
          terminal(Ei),
          valor(Ei, Val, P), !.
  alfabeta_min(Ei, Val, P, Alfa, Beta) :-
          inc, P1 is P + 1,
          V0 is 10000,
          findall(Es, op1(Ei, _, Es), Estados),
          processa_lista_min(Estados, P1, V0, Alfa, Beta, Val).

  processa_lista_min([], _, V, _, _, V).
  processa_lista_min([E|R], P, V, A, B, VFinal) :-
          alfabeta_max(E, V2, P, A, B),
          min(V, V2, Vmin),
          (Vmin =< A -> VFinal = Vmin ;
          min(B, Vmin, B1),
          processa_lista_min(R, P, Vmin, A, B1, VFinal)).

  alfabeta_max(Ei, Val, P, _, _) :- %terminal_MAX
          terminal(Ei),
          valor(Ei, Val, P), !.
  ```
  ```

$alfabeta\_max(Ei, Val, P, Alfa, Beta) :-$
$inc, P1\ is\ P\ +\ 1,$
$V0\ is\ -10000,$
$findall(Es, op1(Ei, \_, Es), Estados),$
$processa\_lista\_max(Estados, P1, V0, Alfa, Beta, Val).$

$processa\_lista\_max([], \_, V, \_, \_, V).$
$processa\_lista\_max([E|R], P, V, A, B, VFinal) :-$
$alfabeta\_min(E, V2, P, A, B),$
$max(V, V2, Vmax),$
$(Vmax\ >=\ B\ ->\ VFinal\ =\ Vmax\ ;$
$max(A, Vmax, A1),$
$processa\_lista\_max(R, P, Vmax, A1, B, VFinal)).$

$min(A, B, A) :- A\ =<\ B, !.$
$min(\_, B, B).$
$max(A, B, B) :- A\ =<\ B, !.$
$max(A, \_, A).$

$escolhe\_max([V - Op|Rest], MelhorOp) :-$
$escolhe\_max(Rest, V - Op, MelhorOp).$
$escolhe\_max([], \_ - Op, Op).$
$escolhe\_max([V1 - \_|T], V0 - Op0, MelhorOp) :-$
$V1\ =<\ V0, !, escolhe\_max(T, V0 - Op0, MelhorOp).$
$escolhe\_max([V1 - Op1|T], \_, MelhorOp) :-$
$escolhe\_max(T, V1 - Op1, MelhorOp).$

- Comparação de resultados entre $MinMax$ e $Alfa - Beta$:
  - Consultamos os ficheiros $[sos]$. e $[minmax]$. para realizar a comparação a partir do mesmo $estado\_inicial$.

    $$estado\_inicial\big(e([s, v, v, v, v, s, v, v, o], x)\big).$$

    $|\ ?- inicializa\_contador, estado\_inicial(E),$
    $minimax\_decidir(E, Op), nos\_visitados(N).$
    - Tempo de execução $\approx 2108\ ms$
    - Nós visitados $= 75972\ nós$
  - Consultamos agora os ficheiros $[sos]$. e $[alfabeta]$. para realizar a comparação a partir do mesmo $estado\_inicial$.

    $$estado\_inicial\big(e([s, v, v, v, v, s, v, v, o], x)\big).$$

    $|\ ?- inicializa\_contador, estado\_inicial(E),$
    $alfabeta(E, Op), nos\_visitados(N).$
    - Tempo de execução $\approx 21\ ms$
    - Nós visitados $= 3090\ nós$

- Concluindo, após analisar a eficácia de ambos os algoritmos, verificamos que ambos os algoritmos produzem uma jogada válida, no entanto, o algoritmo $Alfa - Beta$ permite encontrar uma solução com uma redução bastante significativa de recursos computacionais em comparação com o algoritmo $MinMax$.

Alínea f)

Alínea g)

- $joga :-$

```
joga :-
        [alfabeta], %[minmax],
        estado_inicial(Ei), printTabuleiro(Ei),
        jogar_loop(Ei, x).

jogar_loop(Estado, Jogador) :-
        (terminal(Estado) -> write('Fim do jogo!'), nl,
                valor(Estado, V, 0),
                format('Resultado: ~w~n', [V]),
                printTabuleiro(Estado);
                (Jogador = x -> % Humano joga
                        format('Jogador ~w, escolha a
                                posição (1 - 9): ~n', [Jogador]),
                        read(Pos),
                        format('Jogador ~w, escolha a
                                letra (s ou o): ~n', [Jogador]),
                        read(Letra),
                        (op1(Estado, joga(Pos, Letra), NovoEstado) ->
                                printTabuleiro(NovoEstado),
                                inv(Jogador, ProximoJogador),
                                jogar_loop(NovoEstado, ProximoJogador);
                                write('Jogada inválida, tenta outra vez.'), nl,
                                jogar_loop(Estado, Jogador))

                        ;

                        alfabeta(Estado, Op), %Agente joga (Jogador = y)
                        %minimax_decidir(Estado, Op),
                        format('Agente escolhe: ~w~n', [Op]),
                        op1(Estado, Op, NovoEstado),
                        printTabuleiro(NovoEstado),
                        inv(Jogador, ProximoJogador),
                        jogar_loop(NovoEstado, ProximoJogador)
                )
        ).
```

$printTabuleiro(e([A,B,C,D,E,F,G,H,I],Jogador)) :-$
$format('\sim w \mid \sim w \mid \sim w \sim n',[A,B,C]),$
$format('\sim w \mid \sim w \mid \sim w \sim n',[D,E,F]),$
$format('\sim w \mid \sim w \mid \sim w \sim n',[G,H,I]),$
$format('Jogador\ atual: \sim w \sim n \sim n',[Jogador]).$

Alínea h)

- Usamos o estado inicial da *alínea a)*, onde o tabuleiro contém 9 casas livres. Expandimos a árvore até profundidade 3, onde ainda conseguimos realizar os cálculos de forma relativamente fácil, visto que apenas nesta última profundidade existem estados terminais. O número de nós expandidos será idêntico para os algoritmos $Minimax$ e $Alfa - Beta$. No entanto, o número de nós avaliados pela função de utilidade será inferior no Alfa-Beta, devido aos cortes efetuados com base nos valores mínimo e máximo parciais, permitindo ao algoritmo ignorar ramos que não influenciam a decisão ótima.

	Número de Nós Expandidos			
	Prof.1	Prof. 2	Prof. 3	Total
$MinMax$	18	36	72	126
$Alfa - Beta$	18	36	72	126

Exercício 2

Alínea a)

- $\%estado\_inicial\big(e([Tabuleiro, SOSdex, SOSdey, PeçasJogadas], Jogador)\big).$
  $estado\_inicial(e([[v,v,v,v,v,v,v,v,v], SOSdex, SOSdey,$
  $pecas], jogador)).$
- $inv(x,y).$
  $inv(y,x).$

Alínea b)

- $\%Estado\_terminal$
  $terminal(e([\_,\_,\_,9],\_)).$

Alínea c)

- $\%Funcao\_utilidade$

  $valor(e([\_,Px,Py,\_],\_), \quad 1,\_) :- Px > Py, !.$

  $valor(e([\_,Px,Py,\_],\_), -1,\_) :- Px < Py, !.$

  $valor(e([\_,Px,Py,\_],\_), \quad 0,\_) :- Px =:= Py.$

Alínea d)

Alínea e)

- Usamos o estado inicial da *alínea a)*, onde o tabuleiro contém 9 casas livres. Expandimos a árvore até profundidade 3, onde ainda conseguimos realizar os cálculos de forma relativamente fácil, visto que apenas nesta última profundidade existem estados terminais. O número de nós expandidos será idêntico para os algoritmos $Minimax$ e $Alfa - Beta$. No entanto, o número de nós avaliados pela função de utilidade será inferior no Alfa-Beta, devido aos cortes efetuados com base nos valores mínimo e máximo parciais, permitindo ao algoritmo ignorar ramos que não influenciam a decisão ótima.

	Número de Nós Expandidos			
	Prof.1	Prof. 2	Prof. 3	Total
$MinMax$	18	36	72	126
$Alfa - Beta$	18	36	72	126