UNIVERSIDADE DE ÉVORA

Engenharia Informática

Inteligência Artificial

Terceiro Trabalho

Autores:

Gabriel Charrua 32457
João Silva 32355

Docente:

Irene Pimenta Rodrigues

Introdução

Antes de mais, a realização deste trabalho encontra-se inserida na cadeira de Inteligência Artificial mais especificamente na componente prática. A realização deste trabalho foi proposta pela docente Irene Pimenta Rodrigues. Em relação à cadeira, encontra-se inserida na Licenciatura em Engenharia Informática da Universidade de Évora, 6º semestre.

Em relação ao propósito deste trabalho, pretende-se fazer o "Jogo do Galo" e um jogo à nossa escolha. Ambos os jogos vão ter por objectivo cumprir um determinado conjunto de regras e restrições para que corram com os algoritmos dados nas aulas teóricas e nas aulas práticas. Tal como referimos anteriormente, além do "Jogo do Galo" temos de escolher outro e jogo; nós, como segundo jogo, escolhemos o "Quatro em Linha". A escolha do segundo jogo está relacionada com a quantidade de informação que recolhemos sobre ele sendo que também estivemos interessados no "Ouri". O "Jogo do Galo" é conhecido por praticamente todas as pessoas, sendo que essa deve ter sido uma das razões pela qual a docente optou pela sua escolha enquanto que o "Quatro em Linha", não sendo desconhecido, não é tão popular quanto o "Jogo do Galo"; mesmo assim, esperamos conseguir realizar este trabalho o melhor possível e entrega-lo a tempo e horas.

Para finalizar, queríamos dizer que, com o código das aulas práticas, esperamos ter o trabalho bastante facilitado uma vez o tempo que teríamos de dedicar para o fazer podemos utilizá-lo para outros fins.

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

R: A estrutura que escolhemos foi a lista. Em baixo podem ver-se as representações do "Jogo do Galo" e do "Quatro em Linha", respectivamente.

```
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), _)],_)).

estado_inicial(([e,e,e,p,p,p,p,e,p,p,e,e,e,p,e,e,e,p,p,p], _, _, _)).
```

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

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

R: O predicado "terminal(estado)" vai ser igual para ambos os jogos e, por isso, em baixo apenas vamos colocar o exemplo de um deles.

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

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.

R: Neste caso, se o tabuleiro já estiver todo preenchido e não estiverem três em linha, então sabemos que é empate; caso cada jogador já tenho jogado pelo menos três vezes e que um deles tenha conseguido colocar três em linha, então, o último símbolo a ser jogado (o que ganha) é impresso no ecrã.

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

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

R: Em baixo pode ver-se a implementação dada pela professora na aula; e é essa que vamos usar de modo a escolher a melhor jogado num certo estado.

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

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

% se é estado terminal não há jogada
minimax_decidir(Ei,terminou):- terminal(Ei).

%Para cada estado sucessor de Ei calcula o valor minimax do estado
%Opf é o operador (jogada) que tem maior valor

minimax_decidir(Ei,Opf):-
    findall(Es-Op, op1(Ei,Op,Es),L),
    length(L,S),
    incMais(S),
    findall(Vc-Op,(member(E-Op,L), minimax_valor(E,Vc,1)),L1),
    escolhe_max(L1,Opf).

% se um estado é terminal o valor é dado pela função de utilidade
%minimax_valor(Ei,Val,P):- terminal(Ei), valor(Ei,Val,P).
minimax_valor(Ei,Val,P):- terminal(Ei), !, valor(Ei,Val,P).


%Se o estado não é terminal o valor é:
% -se a profundidade é par, o maior valor dos sucessores de Ei
% -se aprofundidade é impar o menor valor dos sucessores de Ei
minimax_valor(Ei,Val,P):-
    findall(Es,op1(Ei,_,Es),L),
    length(L,S),
    incMais(S),
    P1 is P+1,
    findall(Val1,(member(E,L),minimax_valor(E,Val1,P1)),V),
    seleciona_valor(V,P,Val).


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

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

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

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


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).

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).
```

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

R: Como podemos ver, em baixo, tal como nos foi pedido, está o definido a pesquisa Alfa-Beta.

```
alfabeta_decidir(Ei,terminou):- terminal(Ei).

alfabeta_decidir(Ei,Opf):-
    findall(Es-Op, op1(Ei,Op,Es),L),
    length(L, S),
    incMais(S),
    incProf,
    findall(Vc-Op,(member(E-Op,L), alfabeta_valor(E,Vc,2000,-2000)),L1),
    escolhe_max(L1,Opf).

% se um estado é terminal o valor é dado pela função de utilidade
alfabeta_valor(Ei,Val,_,_):- terminal(Ei), !, valor(Ei,Val,_).

alfabeta_valor(Ei,Val,Alfa,Beta):-
    findall(Es,op1(Ei,_,Es),L),
    length(L, S),
    incMais(S),
    incProf,
    corte_min(L,Alfa,Beta,Val,[]).

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

min_valor(Ei,Val,_,_):-
    p(P),
    prof(Lim),
    P>=Lim,
    func_aval(Ei,Val,_),!.

min_valor(Ei,Val,Alfa,Beta):-
    findall(Es,op1(Ei,_,Es),L),
    length(L,M),
    incMais(M),
    corte_max(L,Alfa,Beta,Val,[]).

corte_max([],_,_,Max,Vals):-
    maximo1(Vals,Max).

corte_max(_,Alfa,_,Val,[Val|_]):-
    Val>=Alfa,!.

corte_max([E|Es],Alfa,_,V,Vs):-
    maximo1(Vs,NovoBeta),
    alfabeta_valor(E,Val,Alfa,NovoBeta),
    corte_max(Es,Alfa,NovoBeta,V,[Val|Vs]).
```

```
corte_min([],_,_,Min,Vals):-
    minimo1(Vals,Min).

corte_min(_,_,Beta,Val,[Val|_]):-
    Val=<Beta,!.

corte_min([E|Es],_,Beta,V,Vs):-
    minimo1(Vs,NovoAlfa),
    min_valor(E,Val,NovoAlfa,Beta),
    corte_min(Es,NovoAlfa,Beta,V,[Val|Vs]).

maximo1([],-2000).
maximo1([A|R],Val):- maximo1(R,A,Val).

maximo1([],A,A).
maximo1([A|R],X,Val):- A < X,!, maximo1(R,X,Val).
maximo1([A|R],_,Val):-  maximo1(R,A,Val).


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).

minimo1([],2000).
minimo1([A|R],Val):- minimo1(R,A,Val).

minimo1([],A,A).
minimo1([A|R],X,Val):- A > X,!, minimo1(R,X,Val).
minimo1([A|R],_,Val):-  minimo1(R,A,Val).
```

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.

R: Antes de mais, a vamos colocar a built-in que usámos (statistics), e o que interessa saber; de seguida, como será visível, vai estar a função que é praticamente a mesma para os dois jogos (vamos meter só uma) e os resultados.

statistics(*+Key, -Value*)

Unify system statistics determined by *Key* with *Value*. The possible keys are given in the table 6. This predicate supports additional keys for compatibility reasons. These keys are described in table 7.

real_time	[Wall time, Wall time since last] (integer seconds. See get_time/1)

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

ciclo_jogada('c',(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),
    ciclo_jogada('j',Es).

ciclo_jogada('j',(E,J)):-
    print_(E),
    nl,
    write('Escreva a linha da posicao onde deseja jogar: '),
    read(X),
    write('Escreva a coluna da posicao onde deseja jogar: '),
    read(Y),
    inverteJog(J,J1),
    op1((E,J),insere(p(X,Y),J1),Es),
    ciclo_jogada('c',Es).
```

```
Tempo: (26221)
Numero de nos: (549945)
insere(p(3,3),o)
```

```
Tempo: (687)
Numero de nos: (1407)
insere_coluna(7,o)
```

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

R: O agente inteligente pode ver-se no print anterior em "**ciclo_jogada('j', (E, J))**" e este agente o que faz é preencher, utilizando o algoritmo "minimax" ou o algoritmo "alfa-beta", a casa que impede o humano de ganhar e, ao mesmo tempo, aumenta as suas hipóteses de sucesso.

8. Apresente uma tabela com o número de nós expandidos para diferentes estados dos 2 jogos (10 estados de cada jogo no mínimo) com os vários algoritmos.

R: Podemos ver a baixo o número de nós expandidos primeiro para o galo (com minimax) e depois para o jogo quatro em linha (alfa-beta).

Galo	**Quatro em Linha**
1- 59704	**1-** 1298
2- 1052	**2-** 124
3- 46	**3-** 104
4- 4	**4-** 70

Conclusão

Em primeiro lugar, com a realização deste trabalho, sentimos que estamos mais preparados para as avaliações que destas matérias resultarem. Neste segundo trabalho sentimos bastantes mais dificuldades do que no primeiro; não só pelo tempo que foi bastante curto como também pelos trabalhos e testes de outras unidades curriculares a que estivemos e vamos estar sujeitos durante este período.

Em relação ao trabalho em si, conseguimos completar quase todas as tarefas que nos foram pedidas. No "Jogo do Galo", conseguimos correr o jogo com o algoritmo "minimax", enquanto para o correr com o algoritmo alfa-beta temos de fazer umas pequenas alterações; já o "Quatro em Linha", temos de fazer umas pequenas alterações para o correr com o algoritmo "minimax" e está a correr com o "alfa-beta". No que toca às questões colocadas no enunciado do trabalho, conseguimos responder a todas elas mesmo que em algumas tenhamos tido algumas dúvidas ao longo do trabalho; parte dessas dúvidas foram dissipadas, enquanto outras permanecem. No entanto, sabemos que na apresentação do trabalho a docente nos irá corrigir e ficaremos com todas as dúvidas esclarecidas.

Para concluir, como já referimos anteriormente, para além da nota, este trabalho também tem por objectivo preparar os alunos para outros elementos de avaliação o que, pessoalmente, achamos que foi bem conseguido (ainda que não bem conseguido como no primeiro).