Manual Técnico

Inteligência Artificial | Projeto 2 - Jogo do Cavalo (2 jogadores) | André Meseiro 202100225 e Pedro Anjos 202100230

1. Algoritmo Implementado, devidamente comentado e respetivas funções auxiliares

1.1. Função NEGAMAX com cortes alfabeta

```
;; Função negamax com cortes alfabeta que recebe um nó, uma profundidade, um alfa,
um beta, um jogador, uma lista de jogadores, uma função de sucessores e uma função
de avaliação
;; e retorna o valor do nó
(defun negamax (no profundidade alfa beta jogador jogadores funcao-sucessores
funcao-avaliacao &optional (cache (make-hash-table)) (limite 5000) (folga 50)
(tempo-inicial (get-internal-real-time)))
  "Função negamax com cortes alfabeta"
  (let ((sucessores (funcall funcao-sucessores no jogador)))
    (setf *nos-gerados* (+ *nos-gerados* (length sucessores)))
    (setf *nos-expandidos* (+ *nos-expandidos* 1))
    (cond ((or (zerop profundidade))
           (list no (* (cond ((equal jogador (first jogadores)) 1) (t -1))
(procurar-cache (no-estado no) jogador funcao-avaliacao cache))))
          (t (cond ((null sucessores) nil)
                   (t (negamax-auxiliar no sucessores profundidade alfa beta
jogador jogadores funcao-sucessores funcao-avaliacao cache limite folga tempo-
inicial))
              )
          )
        )
    )
)
```

1.2. Função auxiliar do algoritmo NEGAMAX

```
;; Função auxiliar do negamax
(defun negamax-auxiliar (no sucessores profundidade alfa beta jogador jogadores
funcao-sucessores funcao-avaliacao cache limite folga tempo-inicial &aux (tempo-
atual (get-internal-real-time)))
  "Função auxiliar do negamax"
  (cond ((null sucessores) nil)
        ((>= (* (/ (- tempo-atual tempo-inicial) internal-time-units-per-second)
1000) (- limite folga)) nil)
        (t (let* ((valor (inverter-sinal-utilidade (negamax (car sucessores) (1-
profundidade) (- beta) (- alfa) (trocar-jogador jogador jogadores) jogadores
funcao-sucessores funcao-avaliacao cache limite folga tempo-inicial))))
                (cond ((>= (max alfa (cond ((null (second valor)) most-negative-
double-float) (t (second valor)))) beta) (progn (setf *cortes-realizados* (+
```

```
*cortes-realizados* 1)) nil))
                    (t (no-max-utilidade (append (list valor) (list (negamax-
auxiliar no (cdr sucessores) profundidade (max alfa (cond ((null (second valor))
most-negative-double-float) (t (second valor)))) beta jogador jogadores funcao-
sucessores funcao-avaliacao cache limite folga tempo-inicial))) jogador))
              )
            )
          )
  )
)
```

1.3. Função alfabeta

```
;; Função alfa-beta que executa o negamax com cortes alfa-beta e retorna o valor
do nó e o tempo de execução
(defun alfabeta (no profundidade alfa beta jogador jogadores funcao-sucessores
funcao-avaliacao &optional (cache (make-hash-table)) (limite 5000) (folga 10)
(tempo-inicial (get-internal-real-time)))
  "Função alfa-beta que executa o negamax com cortes alfa-beta e retorna o valor
do nó e o tempo de execução"
  (setf *nos-gerados* 0)
  (setf *nos-expandidos* 0)
  (setf *cortes-realizados* 0)
  (let* ((jogada-calculada (negamax no profundidade alfa beta jogador jogadores
funcao-sucessores funcao-avaliacao cache limite folga tempo-inicial))
         (jogada (jogada-a-realizar no (first jogada-calculada))))
    (list (no-estado jogada) (no-profundidade (first jogada-calculada)) (second
jogada-calculada) (/ (- (get-internal-real-time) tempo-inicial) internal-time-
units-per-second) *nos-expandidos* *nos-gerados* *cortes-realizados*)
  )
)
```

1.4. Função procura-cache

```
;; Função que procura na cache
(defun procurar-cache (estado jogador funcao-avaliacao cache)
  "Função que procura na cache"
  (let ((utilidade (gethash estado cache)))
    (cond ((null utilidade)
           (let ((nova-utilidade (funcall funcao-avaliacao estado jogador)))
             (setf (gethash estado cache) nova-utilidade)
             nova-utilidade
           )
         )
         (t utilidade)
    )
  )
)
```

1.5. Função utilidade

```
;; Função que retorna o nó com a maior utilidade e a sua utilidade
(defun no-max-utilidade (lista jogador &optional max-util)
  "Função que retorna o nó com a maior utilidade e a sua utilidade"
    (cond ((null lista) max-util)
        ((null (car lista)) (no-max-utilidade (cdr lista) jogador max-util))
        ((null max-util) (no-max-utilidade (cdr lista) jogador (car lista)))
        ((> (second (car lista)) (second max-util)) (no-max-utilidade (cdr lista)
jogador (car lista)))
        (t (no-max-utilidade (cdr lista) jogador max-util))
  )
)
```

1.6. Função inverte sinal da utilidade

```
;; Função que inverte o sinal da utilidade
(defun inverter-sinal-utilidade (utilidade)
  "Função que inverte o sinal da utilidade"
  (cond ((null utilidade) nil)
        (t (list (first utilidade) (- (second utilidade))))
  )
)
```

1.7. Função troca-jogador

```
;; Função que altera o jogador atual.
(defun trocar-jogador (jogador-atual jogadores)
  "Função que altera o jogador atual"
  (cond ((equal jogador-atual (first jogadores)) (second jogadores))
        ((equal jogador-atual (second jogadores)) (first jogadores))
  )
)
```

1.8. Função jogada a realizar

```
;; Função que recebe o nó atual, o nó que foi caculado e retorna a jogada a
realizar.
(defun jogada-a-realizar (no-atual no-calculado)
  "Função que retorna a jogada a realizar"
  (cond ((or (null no-atual) (null no-calculado)) nil)
        ((equal no-atual (no-pai no-calculado)) no-calculado)
        (t (jogada-a-realizar no-atual (no-pai no-calculado)))
  )
)
```

2. Descrição dos tipos abstratos usados no programa

- Nó - Objeto base do projeto; Consiste numa lista que contém o tabuleiro, o custo e o nó pai (antecessor);
- Estado - Consiste no estado atual do tabuleiro juntamente com uma lista que contém as pontuações dos dois jogadores;
- Operador - Consiste numa operação a aplicar a um nó; Neste contexto está relacionado com uma operação intermédia necessária para chegar ao estado pretendido, associado à jogada efetuada;
- Sucessores - Consiste numa lista que contém os nós sucessores de um determinado nó, após a aplicação de um operador; Neste contexto, diz respeito a todas as casas disponíveis para a jogada seguinte;
- Resultados - Consistem em novos estados que são retornados juntamente com estatísticas.

3. Limitações

No que diz respeito às limitações, mais para o final dos jogos, existem jogadas que são possíveis de fazer, mas que acabam por não ser realizadas, por parte do computador.

4. Análise critica dos resultados das execuções do programa, transparecendo a compreensão das limitações do projeto

Devido às limitações mencionadas no ponto anterior, os jogos acabam sempre por terminar, com um vencedor, mas ainda existem jogadas por fazer por parte do computador, para os resultados obtidos.

5. Análise estatística acerca de uma execução do programa contra um adversário humano (incluindo limite de tempo usado e, para cada jogada: o respetivo valor, a profundidade do grafo de jogo e o número de cortes efetuado no processo de análise, com recurso ao ficheiro log.dat)

Limite de tempo: 2.5 segundos

Jogada	Valor utilidade	Profundidade	Cortes
1	95	10	1250
2	-81	10	1362
3	-69	10	1446
4	-100	10	1538
5	-11	10	1687
6	47	10	748
7	-117	10	1366
8	24	10	1298

Jogada	Valor utilidade	Profundidade	Cortes
9	106	10	2102
10	115	10	1928
11	134	10	2690
12	163	10	2144
13	166	10	1924
14	0	10	796
15	87	10	828
16	38	10	72
17	41	10	7
18	90	10	0
19	45	10	0