Manual Técnico

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal

2020/2021

Problema do Quatro

Realizado por:

André Nascimento nº160221075

Eduardo Ferreira nº110221031

Arquitetura do sistema

Para melhor organização do código correspondente ao problema, foram criados 3 ficheiros:

Algoritm.LISP - Implementa os algoritmos relevantes para a resolução do problema.

Interact.LISP - É a pagina de interação do utilizador.

Jogo.LISP - Contem o jogo em si, juntamente como as peças são colocadas.

Entidades e a sua implementação

Como entidade o nosso negamax tem o Node, que basicamente é o estado atual do jogo.

Tambem temos a entidade depth, que será a profundidade maxima ao descer.

Tambem temos a entidade player, que simboliza qual o jogador que vai efetuar a jogada do nosso sucessor (alternando entre valores de 1 e -1).

Tambem temos a entidade time, que simboliza o tempo que o algoritmo tem para executar a jogada.

Temos tambem o alfa, onde é inicialmente atribuido um número infinito negativo.

Temos tambem o beta, onde é inicialmente atribuido um número infinito positivo.

Temos tambem o moves, onde vão ser guardadas as jogadas efetuadas.

Temos tambem o start_time, que vai servir como cronómetro e quando o tempo chegar ao atributo time que nós temos, o algoritmo fecha.

Para alem desse temos o reverse_counter, que irá servir como condição de paragem final e como forma de segurança para se certificar que todos os nós do gráfo foram avaliados/cortados e termina mesmo no fim.

Algoritmos e a sua implementação

O algoritmo implementado neste projeto foi:

- Negamax (com fail-soft)

Negamax:

```
(defun boardNPieces ()
;;"Representa um tabuleiro de teste"
  '((((branca quadrada alta cheia) 0 (branca quadrada baixa oca) 0) ((preta quadrada alta cheia) 0 0 0) (0 0 0 0) (0 0 0 0))
(
(branca quadrada alta oca)

(branca quadrada baixa cheia)

(preta quadrada alta oca)
(preta quadrada baixa oca)
(preta quadrada baixa cheia)
(branca redonda alta cheia)
(branca redonda alta oca)
(branca redonda baixa cheia)
(branca redonda baixa oca)
(preta redonda alta cheia)
(preta redonda alta oca)
(preta redonda baixa cheia)
(preta redonda baixa oca)
))
)
```

Figura 1: Imagem do teste 1.

```
CL-USER 3 : 1 > (negamax (boardNPieces) 4 1 5)
(((BRANCA QUADRADA ALTA OCA) (3 0)))

CL-USER 4 : 1 > (get-board-ready-for-ristic '((BRANCA QUADRADA ALTA OCA) (3 0)) (boardNPieces))
15

CL-USER 5 : 1 > (minimax (boardNPieces))
15
```

Figura 2: Resultados do teste 1.

```
;;;; Projeto de IA 1o fase
;;;; Disciplina de IA - 2020 / 2021
;;;; Autores: Eduardo Ferreira, Andre Nascimento

(defun boardNPieces ()
;;"Representa um tabuleiro de teste"
  '((((branca quadrada alta cheia) 0 (branca quadrada baixa oca) (branca quadrada alta oca)) ((preta quadrada alta cheia) (preta quadrada alta oca) 0 0) (0 0 0
(


(branca quadrada baixa cheia)


(preta quadrada baixa oca)
(preta quadrada baixa cheia)
(branca redonda alta cheia)
(branca redonda alta oca)
(branca redonda baixa cheia)
(branca redonda baixa oca)
(preta redonda alta cheia)
(preta redonda alta oca)
(preta redonda baixa cheia)
(preta redonda baixa oca)
))
)
```

Figura 3: Imagem do teste 2

```
CL-USER 10 : 1 > (negamax (boardNPieces) 4 1 5)
(((BRANCA QUADRADA BAIXA CHEIA) (0 1)))

CL-USER 11 : 1 > (get-board-ready-for-ristic '((BRANCA QUADRADA BAIXA CHEIA) (0 1)) (boardNPieces))
1000

CL-USER 12 : 1 > (minimax (boardNPieces))
1000
```

Figura 4: Resultados do teste 2

```
;;///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
;;/////////////////////////////////////////////Ignorar, foi tirado da wikipedia para testar se ambos acham o mesmo valor final como forma a testar se on nosso
;;///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
;;(minimax (boardNPieces))
(defun minimax(node &optional (player t) (moves nil) (depth 3)  (alpha most-negative-fixnum) (beta most-positive-fixnum) (tempo-limite 4000) (start-time (get-u
(cond
((or (= depth 0) (>= (- (get-universal-time) start-time) tempo-limite)) (get-board-ready-for-ristic moves node))

((equal player T)
 (let* ((maxEval most-negative-fixnum))
   (dolist (cunny (runThroughPieces node))
     (setf maxEval (max maxEval (minimax node (not player) cunny (1- depth) alpha beta (- (get-universal-time) start-time))))
     (setf alpha (max alpha maxEval))
     (when (>= alpha beta)
            (return))
   )
  maxEval)
)

(t
 (let* ((minEval most-positive-fixnum))
   (dolist (cunny (runThroughPieces node))
      (setf minEval (min minEval (minimax node (not player) cunny (1- depth) alpha beta (- (get-universal-time) start-time))))
      (setf beta (min beta minEval))
      (when (<= beta alpha)
             (return))
   )
   minEval)
)
))
;;///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
;;/////////////////////////////////////////////Ignorar, foi tirado da wikipedia para testar se ambos acham o mesmo valor final como forma a testar se on nosso
;;///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

Figura 5: Imagem do algoritmo minmax

Este algoritmo minimax não é para ser avaliado no projeto, foi simplesmente algo que o grupo encontrou na wikipédia e utilizou como forma de testar se o nosso algoritmo negamax conseguiria achar jogadas que tivessem o mesmo valor heuristico que um algoritmo que o grupo tem a certeza está a funcionar perfeitamente.

Como se pode verificar nas imagens, a peça a ser jogada que o nosso negamax recomenda, assim como as suas coordenadas, acaba por ter o mesmo valor que o minimax encontra.

Estes testes foram efetuados ao alterar o tabuleiro de teste e colocando determinadas peças previamente no tabuleiro para ver como o nosso negamax iria reagir.

Descrição das opções tomadas

Como entidade o nosso negamax tem o Node, que basicamente é o estado atual do jogo.

Tambem temos a entidade depth, que será a profundidade maxima ao descer.

Tambem temos a entidade player, que simboliza qual o jogador que vai efetuar a jogada do nosso sucessor (alternando entre valores de 1 e -1).

Tambem temos a entidade time, que simboliza o tempo que o algoritmo tem para executar a jogada.

Temos tambem o alfa, onde é inicialmente atribuido um número infinito negativo.

Temos tambem o beta, onde é inicialmente atribuido um número infinito positivo.

Temos tambem o moves, onde vão ser guardadas as jogadas efetuadas.

Temos tambem o start_time, que vai servir como cronómetro e quando o tempo chegar ao atributo time que nós temos, o algoritmo fecha.

Para alem desse temos o reverse_counter, que irá servir como condição de paragem final e como forma de segurança para se certificar que todos os nós do gráfo foram avaliados/cortados e termina mesmo no fim.

O algoritmo em si começa com a condição que vai verificar três opções, que são:

- Se a profundidade é 0
- Se a tabuleiro já encontrou a solução
- Se o tempo limite já foi ultrapassado

  Caso alguma destas se verifique, então será devolvido o valor.

  De seguida temos um let com a primeira variavel chamada *sucessors*, que irá encontrar os sucessores de um nó e ordená-los apropriadamente.

  Depois temos a variavel *value*, que se inicia como menos infinito. Esta variavel ser-lhe-à atribuido o valor da função negamax-sucessors, esta função secundária irá servir para simular o foreach/loop que se encontra no powerpoint de teoria de jogos, slide 34.

  Nessa função secundária iremos achar o bestValue que será o max do bestValue com a negação do negamax, após isso, iremos achar o alfa que será o max entre o alfa e o bestValue. Após isso, iremos ter um conjunto de condições para guarantir que o algoritmo corre de uma forma recursiva, essas condições são:

- (>= alfa beta) serão os nossos cortes.
- ((and (null (cdr moves)) (= reverse-counter 1)) moves) será a nossa função termino
- ((null (cdr moves)) value) será a condição de termino para os filhos abaixo dos filhos diretos do nó inicial.

  Após o algoritmo ter sido terminado, irá devolver a melhor jogada que encontrou e caso o utlizador esteja num jogo, será tudo guardado no log.

```
Inicio do jogo:

Profundidade escolhida 3:
Tempo limite escolhido 5

Peca: (BRANCA QUADRADA ALTA CHEIA) Coordenada X: 0 Coordenada Y 0
Tabuleiro ((((BRANCA QUADRADA ALTA CHEIA) 0 0 0) (0 0 0 0) (0 0 0 0) (0 0 0 0)) ((BRANCA QUADRADA ALTA OCA) (BRANCA QUADRADA BAIXA OCA) (BRANCA QUADRADA BAIXA CHEIA) (PRETA QUADRADA ALTA CHEIA) (PRET
BAIXA CHEIA) (BRANCA REDONDA ALTA CHEIA) (BRANCA REDONDA ALTA OCA) (BRANCA REDONDA BAIXA CHEIA) (BRANCA REDONDA BAIXA OCA) (PRETA REDONDA ALTA CHEIA) (PRETA REDONDA ALTA OCA) (PRETA REDONDA BAIXA CHEI


Peca: (BRANCA QUADRADA ALTA OCA) Coordenada X: 0 Coordenada Y 2
Tabuleiro ((((BRANCA QUADRADA ALTA CHEIA) 0 (BRANCA QUADRADA ALTA OCA) 0) (0 0 0 0) (0 0 0 0) (0 0 0 0)) ((BRANCA QUADRADA BAIXA OCA) (BRANCA QUADRADA BAIXA CHEIA) (PRETA QUADRADA ALTA CHEIA) (PRETA
BAIXA CHEIA) (BRANCA REDONDA ALTA CHEIA) (BRANCA REDONDA ALTA OCA) (BRANCA REDONDA BAIXA CHEIA) (BRANCA REDONDA BAIXA OCA) (PRETA REDONDA ALTA CHEIA) (PRETA REDONDA ALTA OCA) (PRETA REDONDA BAIXA CHEI


Peca: (PRETA REDONDA BAIXA OCA) Coordenada X: 0 Coordenada Y 1
Tabuleiro ((((BRANCA QUADRADA ALTA CHEIA) (PRETA REDONDA BAIXA OCA) (BRANCA QUADRADA ALTA OCA) 0) (0 0 0 0) (0 0 0 0) (0 0 0 0)) ((BRANCA QUADRADA BAIXA OCA) (BRANCA QUADRADA BAIXA CHEIA) (PRETA QUAD
OCA) (PRETA QUADRADA BAIXA CHEIA) (BRANCA REDONDA ALTA CHEIA) (BRANCA REDONDA ALTA OCA) (BRANCA REDONDA BAIXA CHEIA) (BRANCA REDONDA BAIXA OCA) (PRETA REDONDA ALTA CHEIA) (PRETA REDONDA ALTA OCA) (PRE


Peca: (BRANCA QUADRADA BAIXA OCA) Coordenada X: 0 Coordenada Y 3
Tabuleiro ((((BRANCA QUADRADA ALTA CHEIA) (PRETA REDONDA BAIXA OCA) (BRANCA QUADRADA ALTA OCA) (BRANCA QUADRADA BAIXA OCA)) (0 0 0 0) (0 0 0 0) (0 0 0 0)) ((BRANCA QUADRADA BAIXA CHEIA) (PRETA QUADRA
OCA) (PRETA QUADRADA BAIXA CHEIA) (BRANCA REDONDA ALTA CHEIA) (BRANCA REDONDA ALTA OCA) (BRANCA REDONDA BAIXA CHEIA) (BRANCA REDONDA BAIXA OCA) (PRETA REDONDA ALTA CHEIA) (PRETA REDONDA ALTA OCA) (PRE


Peca: (BRANCA QUADRADA BAIXA CHEIA) Coordenada X: 1 Coordenada Y 0
Tabuleiro ((((BRANCA QUADRADA ALTA CHEIA) (PRETA REDONDA BAIXA OCA) (BRANCA QUADRADA ALTA OCA) (BRANCA QUADRADA BAIXA OCA)) ((BRANCA QUADRADA BAIXA CHEIA) 0 0 0) (0 0 0 0) (0 0 0 0)) ((PRETA QUADRADA
(PRETA QUADRADA BAIXA CHEIA) (BRANCA REDONDA ALTA CHEIA) (BRANCA REDONDA ALTA OCA) (BRANCA REDONDA BAIXA CHEIA) (BRANCA REDONDA BAIXA OCA) (PRETA REDONDA ALTA CHEIA) (PRETA REDONDA ALTA OCA) (PRETA RE


Peca: (PRETA QUADRADA ALTA CHEIA) Coordenada X: 2 Coordenada Y 0
Tabuleiro ((((BRANCA QUADRADA ALTA CHEIA) (PRETA REDONDA BAIXA OCA) (BRANCA QUADRADA ALTA OCA) (BRANCA QUADRADA BAIXA OCA)) ((BRANCA QUADRADA BAIXA CHEIA) 0 0 0) ((PRETA QUADRADA ALTA CHEIA) 0 0 0) (
(PRETA QUADRADA BAIXA CHEIA) (BRANCA REDONDA ALTA CHEIA) (BRANCA REDONDA ALTA OCA) (BRANCA REDONDA BAIXA CHEIA) (BRANCA REDONDA BAIXA OCA) (PRETA REDONDA ALTA CHEIA) (PRETA REDONDA ALTA OCA) (PRETA RE


Peca: (PRETA QUADRADA ALTA OCA) Coordenada X: 3 Coordenada Y 0
Tabuleiro ((((BRANCA QUADRADA ALTA CHEIA) (PRETA REDONDA BAIXA OCA) (BRANCA QUADRADA ALTA OCA) (BRANCA QUADRADA BAIXA OCA)) ((BRANCA QUADRADA BAIXA CHEIA) 0 0 0) ((PRETA QUADRADA ALTA CHEIA) 0 0 0) (
(PRETA QUADRADA BAIXA CHEIA) (BRANCA REDONDA ALTA CHEIA) (BRANCA REDONDA ALTA OCA) (BRANCA REDONDA BAIXA CHEIA) (BRANCA REDONDA BAIXA OCA) (PRETA REDONDA ALTA CHEIA) (PRETA REDONDA ALTA OCA) (PRETA RE


Fim do Jogo:

O vencedor Foi: -1
```

Figura 6: Imagem do log.

Limitações técnicas

A limitação de Stack ainda existe, mas o comando para aumentar a Stack foi incluido na função *iniciar*, para que não seja preciso executa-lo separadamente.