Inteligência Artificial - Jogo do Cavalo, 2 Jogadores

Manual Técnico

2019/2020

Autores:

- Henoch Mendes Vitureira Nº170221014
- Patrick Battisti Forsthofer Nº190200007

Docentes:

- Prof. Joaquim Filipe
- Prof. Filipe Mariano

Tipos Abstratos Utilizados

Nó

Este tipo abstrado serviu para representar os nós de uma árvore, aplicando o domínio do jogo do cavalo. Estes tinham a estutura: < nó >= pontos da casa atual, pontos do jogador 1, pontos do jogador 2, < estado >, valor heurístico, número do jogador atual. O estado corresponde ao tabuleiro atual do nó. A função abaixo foi utilizada como auxilar à construção de nós.

```
;;;<node>= current-points, points-of-player1, points-of-player2, <state>, f,
player
;;;<state>= current-board
(defun make-node (points p-p1 p-p2 board f player depth)
  "Devolve um novo nó"
  (list points p-p1 p-p2 board f player depth)
)
```

Nó-Jogada

Este tipo abstrado serviu para representar os nós correspondentes a uma jogada a ser aplicada. Estes tinham a estutura: < nó >= < estado >, valor heurístico, valor da casa da jogada, número do jogador atual. O estado corresponde ao tabuleiro após a jogada. A função abaixo foi utilizada como auxilar à construção de nós jogada.

```
;;;<play-node>= <state> f move(cell-points) player
;;;<state>= current-board
```

```
(defun make-play-node (board f move player)
  (list board f move player)
)
```

Algoritmo Implementado: NEGAMAX

O algoritmo implementado para decidir as jogadas do computador foi o NEGAMAX, uma formulação do MINIMAX em que se passa a procurar sempre apenas o máximo, mas se troca o sinal em cada nível, após o backup da árvore gerada. Este recebe um nó, o tempo limite de jogadam, a profundidade atual, o valor de alpha e de beta e o tempo em que a jogada começou. As condições de paregem da função são: Se a profundidade atual é 0 (raiz), se o tempo da jogada já expirou, ou se o nó é terminal. Se as condições de paragem não estiverem a ser cumpridas, é chamada a função que faz o cíclo recursivo da atualização do valor a ser devolvido pelo NEMAGAX. Antes de esta ser invocada, os sucessores do nó atual são gerados e ordenados por valor heurístico de forma decrescente.

```
(defun negamax (node time-limit depth α β cor &optional (play-start-time (get-
internal-real-time)))

  (cond
      ;;se d = 0 ou n é terminal ou tempo terminou ;;return c * valor heuristico
de n
      ((OR (= depth 0)
        (>= (- (get-internal-real-time) play-start-time) time-limit)
          (null (generate-moves node)))
            (* (node-points node) cor))

      (t (let* (
          (successors (sort-f-nodes (generate-moves node)));;sucessores :=
OrderMoves(GenerateMoves(n))
        )
      ;;loop de sucessores
      (successors-loop successors (node-board node) (node-player node) time-limit
depth α β cor play-start-time)
    )
   )
  )
)
```

Loop de Sucessores

A função successors-loop foi a utilizada para fazer o ciclo recursivo da atualização do valor a ser devolvido pelo NEMAGAX. Após verificar se ainda existem sucessores, a variável global do número de nós analisados é incrementada. Logo após isto, é feita uma consulta à hashtable de memoização de pontos para verificar se o estado atual já foi analisado, dependendo do qual é o jogador a jogar no momento. Se a chave (tabuleiro jogador) já existir, o valor que lhe corresponde é devolvido.

Caso o a chave não exista na hashtable, é calculado o máximo entre o melhor valor atual e o valor correspondente à chamada do NEGAMAX com a profundidade decrementada, assim como os valores de alpha e beta permutados, com sinal contrário, e também com a cor do nó oposta. É calculado o valor máximo entre alpha e o melhor valor decrito anteriomente, e este resultado é atribuído a alpha.

Posteriormente é verificado se a profundidade atual é de primeiro nível, se o melhor valor calculado é maior que o valor recebido por parâmetro e se alpha é menor que beta. Se sim, é atualizado o valor global da melhor jogada como um nó jogada com os atributos do nó atual.

Seguidamente, é verificado se o valor de alfa é menor que o de beta, se sim, é adicionada uma chave na hashtable de pontos com o tabuleiro e jogador atual, com o valor do "melhor valor" calculado e é chamda a segunda iteração da função sucessors-loop, de modo a percorrer todos os sucessores existentes. Caso o valor de alpha seja maior do que o de beta, existe uma situação de corte, onde a variável global correspondente ao tipo de corte atual é incrementada e o melhor valor recebido por parâmetro é devolvido.

```
(defun successors-loop (successors board player time depth α β cor play-start-time
&optional (l-best-value most-negative-fixnum)) ;;para cada sucessor nk em
sucessores
  (cond (
        (null successors) l-best-value)
        (t

        ;;verificar se hashtable contem o valor de um estado e devolver caso sim
        (setf *analyzed-nodes* (1+ *analyzed-nodes*))
        (cond ((not (null (gethash (list board player) *memoization-points-
hash*)))
          (gethash (list board player) *memoization-points-hash*)))

          (let* (
            (best-value (max l-best-value (- (negamax (car successors) time (-
depth 1) (- β) (- α) (- cor) play-start-time)))) ;; bestValue := max (bestValue,
-negamax (nk, d-1, -β, - α, -c))
            (α (max α best-value)) ;; α := max (α, bestValue))
          )

         (cond
                  ((and (= depth 1) (> best-value l-best-value) (< α β))
                    (setf *best-play (make-play-node board (node-f(car
successors))
                     (car (value-node (first (position-node player board)) (second
(position-node player board)) (*board)))
                      player))

              ))

          (cond
           ( (< α β)
            (setf (gethash (list board player) *memoization-points-hash*) best-
value)
              (successors-loop (cdr successors) board player time depth α β cor
play-start-time best-value))
```

```
            (t
                (cond ((= cor 1) (setf *alpha-cuts* (1+ *alpha-cuts*))))
                (cond ((= cor -1) (setf *beta-cuts* (1+ *beta-cuts*))))
              l-best-value)
           )
           )
         )
       ))
```

Funções Auxiliares

Generate Moves

Esta função serviu para gerar os sucessores de um dado nó. Dado um nó, verifica todos os movimentos possíveis que um jogador pode fazer no estado em que se encontra, e por cada movimento possível, devolve uma lista de nós sucessores.

```
(defun generate-moves (node)
  "Geração de nós sucessores de um nó"
  (let* (
        (line-column (position-node (node-player node) (node-board node)))
        (line-index (first line-column))
        (column-index (second line-column))
        (board-no-player (replace-value line-index column-index (node-board node)
nil))
        )

        (cond ((null board-no-player) '())
        (t (append
          (move-avaliable (- line-index 2) (- column-index 1) board-no-player
(node-player node) (node-points-p1 node) (node-points-p2 node)  (node-depth node))
          (move-avaliable (- line-index 2) (+ column-index 1) board-no-player
(node-player node) (node-points-p1 node) (node-points-p2 node)  (node-depth node))
          (move-avaliable (+ line-index 2) (- column-index 1) board-no-player
(node-player node) (node-points-p1 node) (node-points-p2 node)  (node-depth node))
          (move-avaliable (+ line-index 2) (+ column-index 1) board-no-player
(node-player node) (node-points-p1 node) (node-points-p2 node)  (node-depth node))
          (move-avaliable (- line-index 1) (- column-index 2) board-no-player
(node-player node) (node-points-p1 node) (node-points-p2 node)  (node-depth node))
          (move-avaliable (- line-index 1) (+ column-index 2) board-no-player
(node-player node) (node-points-p1 node) (node-points-p2 node)  (node-depth node))
          (move-avaliable (+ line-index 1) (- column-index 2) board-no-player
(node-player node) (node-points-p1 node) (node-points-p2 node)  (node-depth node))
          (move-avaliable (+ line-index 1) (+ column-index 2) board-no-player
(node-player node) (node-points-p1 node) (node-points-p2 node)  (node-depth node))
          ))
        )
    )
)
```

Move Avaiable

Esta função serve de auxiliar à "generate-moves". Recebe um alinha, uma coluna, um estado (tabuleiro), o número identificador do jogador, o número de pontos atuais dos dois jogadores e a profundidade do nó antecessor. Com isto, verifica se é possível realizar uma jogada para a posição correspondente à linha e coluna recebida. Se for possível efetuar uma jogada para a posição em questão, é verificado se já existe em memoização o valor eurístico do nós atual, recebendo a chave "(tabuleiro, jogador)", caso exista, o nó sucessor é devolvido com o valor heurístico presente na hashtable. Caso o valor heurístico não esteja definido na hashtable, o mesmo valor é adicionado com a chave a apropriada e o valor heurístico do nó a devolver é calculado com recurso à função de avaliação.

```
(defun move-avaliable (line-index column-index board player points-1 points-2
depth)
   (cond
   ((or (< line-index 0) (> line-index 9)) nil)
   ((or (< column-index 0) (> column-index 9)) nil)
   ((value-node line-index column-index board)

        (let* (
                        (value (car (value-node line-index column-index board)))
                        (points (sum-move player value points-1 points-2))
                        )
      (cond
      ((not (null (gethash (list board player) *memoization-hash*)))
        (list
          (make-node value (first points) (second points) (replace-value line-
index column-index board player) (gethash (list board player) *memoization-hash*)
player (1+ depth))
        )
      )
        (t
          (setf (gethash (list board player) *memoization-hash*) (evaluate-node
(first points) (second points) player))

          (list
            (make-node value (first points) (second points) (replace-value line-
index column-index board player) (evaluate-node (first points) (second points)
player) player (1+ depth))
          )
       ))))))
```

Reset Varaibles

Esta função serviu para definir as variáveis globais por defeito do modo jogo computador contra computador. Estas foram a tabuleiro, o número do jogador, e os pontos de ambos os jogadores.

```
 ;;resetar as váriaveis apos cada partida
  (defun reset-variables ()
    (setf *board (tabuleiro-aleatorio))
    (setf *player -1)
    (setf *points-1 0)
    (setf *points-2 0)
  )
```

Sum Points

A função abaixo serviu para somar os pontos da casa para onde o jogador atual se moveu ao, à sua totalidade de pontos atuais.

```
(defun sum-points (move)
   (cond
     ((eq *player -1) (setf *points-1 (+ *points-1 move)))
     (t (setf *points-2 (+ *points-2 move)))
   ))
```

Jogar

A função "jogar" foi a utilizada para realizar a jogada do computador num dado turno. É nela que é chamado o algorítmo NEGAMAX, de modo a decidir a jogada do mesmo. A função recebe o tabuleiro atual, o tempo limite de jogada e a profundidade máxima, definida pelo utilizador. O resultado do NEGAMAX é guardada na variável de closure "move". Isto, devido a limitações tecnicas, é o valor da casa para onde se vai jogar. Posteriormente a isto, é obtida a posição do valor da casa devolvida pelo NEGAMAX no tabuleiro, e feito o update ao tabuleiro com a jogada. Caso as posições da jogada/valor da casa forem nulas, é atribuido o valor máximo da primeira linha, se for o primeiro jogador, ou o valro máximo da casa da última linha da tabuleiro. Isto acontece quando se trata da primeira jogada de um computador. Para finalizar, os pontos da jogada são somados ao total de pontos do jogador atual, as estatísticas da jogada são escritas no ficheiro de logs e o tabuleiro atualizado é devolvido.

```
(defun jogar (board time depth)
 "Calls the negamax algorithm to decide the computer's play and returns the
updated board with the move done"
    (setf *beta-cuts* 0)
    (setf *alpha-cuts* 0)
    (setf *analyzed-nodes* 0)
    (setf *best-play (make-play-node *board most-negative-fixnum most-negative-
fixnum -5))

    (let* (
      (play-start-time (get-internal-real-time));;start-time in miliseconds
      (move (negamax (make-node -5 *points-1 *points-2 board -5 *player 0) time
depth most-negative-fixnum most-positive-fixnum 1 play-start-time))
```

```
      (best-move (play-node-move *best-play))
      (indexes-move (position-node move board))
      (indexes-player (position-node *player board))
      (board-with-updated-last-move (replace-value (first indexes-player) (second
indexes-player) board nil))
      (board-with-updated-player-position (replace-value (first indexes-move)
(second indexes-move) board-with-updated-last-move *player))
      )

      (cond
        ((and (null (first indexes-player)) (null (second indexes-player)))
          (setf move (max-first-move-value *player *board))
          (setf indexes-move (position-node move board))
          (setf board-with-updated-player-position (replace-value (first indexes-
move) (second indexes-move) *board *player))
        )
      )

      (cond
        ((not (null board-with-updated-player-position))
          (display-computer-move *player (position-indexes-to-chess indexes-move)
board-with-updated-player-position)

          (sum-points move)

          (write-log board-with-updated-player-position *points-1 *points-2 (get-
play-time-milisecs play-start-time) *alpha-cuts* *beta-cuts* *analyzed-nodes*)
          board-with-updated-player-position
        )
        (t
          *board)
        )
    )
  )
```

Game CC (Computer vs Computer)

Esta função serviu para inicializar um jogo do modo computador contra computador. Faz o reset das variáveis globais e apresenta o tabuleiro atual. A seguir, realiza as jogadas de cada jogador alternadamente até nenhum deles poder jogar, invocando a função "jogar" descrita anteriormente. No final, apresenta a pontuação dos dois jogadores.

```
(defun game-cc (time depth)
    (reset-variables)

    (display-start-board *board)
    (loop while (or (not (null (generate-moves (make-node -5 *points-1 *points-2
*board -5 *player 0))))
      (not (null (generate-moves (make-node -5 *points-1 *points-2 *board -5
```

```
(opposite *player) 0)))) ;;se pelo menos 1 player n pode jogar
        (equal 100 (length (remove-nil-value *board))))
      do

      (setf *board (jogar *board time depth))
      (setf *player (opposite *player))
    )

    (display-points-players *points-1 *points-2)
  )
```

Game HC (Human vs Computer)

À semelhança da função "game-cc", serve para inicializar um jogo do modo humano contra computador. Faz o reset das variáveis globais e apresenta o tabuleiro atual. A seguir, realiza as jogadas de cada jogador alternadamente até nenhum deles poder jogar, invocando a função "human-play" para efetuar a jogada do jogardor humano, e a função "jogar" descrita anteriormente, para efetuar jogada do computador. No final, apresenta a pontuação dos dois jogadores.

```
;;human = -1
;;computer = -2
  (defun game-hc (time depth &optional (first-player -1))
    (reset-variables)

    (setf *player first-player)
    (display-start-board *board)

    (loop while (or (not (null (generate-moves (make-node -5 *points-1 *points-2
*board -5 *player 0))))
        (not (null (generate-moves (make-node -5 *points-1 *points-2 *board -5
(opposite *player) 0))))
          (equal 100 (length (remove-nil-value *board))))
      do

      (cond
        ((equal *player -1)
            (let* (
              (moves (list-points (generate-moves (make-node -5 *points-1 *points-
2 *board -5 *player 0))))
              (moves-avaliable (values-to-chess moves *board))
            )

            (cond
            ((null moves-avaliable)
            ;;adicionar tempo da jogada do humano?????
              (setf *board (human-play (read-play (values-to-chess (remove-point-
list-nil (car *board)) *board)))))
            (t (setf *board (human-play (read-play moves-avaliable))))
            )
```

```
          )

          (terpri)
          (display-board *board)
        )
        (t (setf *board (jogar *board time depth)))
      )

      (setf *player (opposite *player))
    )

    (display-points-players *points-1 *points-2)
  )
```

Human Play

A função "human-play" ocupou-se de efetuar a jogada de um jogador humano. Recebe uma posição de xadres no formato "coluna-linha", no formato convencional de xadrez. Recebendo esta posição, coloca o cavalo do jogador humano no tabuleiro atual, soma os pontos correspondentes, e devolve o tabuleiro atualizado.

```
;;(human-play "I3")
(defun human-play (position)
    (let* (
      (move (car (value-node (first (position-chess-to-indexes position)) (second
(position-chess-to-indexes position)) *board)))
      (indexes-move (position-node move *board))
      (indexes-player (position-node *player *board))
      (board-with-updated-last-move (replace-value (first indexes-player) (second
indexes-player) *board nil))
      (board-with-updated-player-position (replace-value (first indexes-move)
(second indexes-move) board-with-updated-last-move *player))
      )

      (cond
      ((and (null (first indexes-player)) (null (second indexes-player)))
        (setf indexes-move (position-node move *board))
        (setf board-with-updated-player-position (replace-value (first indexes-
move) (second indexes-move) *board *player))
      )
    )

      (sum-points move)
      board-with-updated-player-position
      )
  )
```

Write Log

A função utilizada para escrever as estatísticas de uma jogada foi a "write-log". Recebe o tabuleiro de uma jogada, os pontos de cada jogador, o tempo da jogada, o número de cortes de cada tipo (alpha e beta) e o número de nós analisados. A informação recebida é escrita de forma legível no ficheiro de logs "log.dat", com recurso à função "with-open-file".

```
(defun write-log (board points-1 points-2 play-time alpha-cuts beta-cuts num-
analized-nodes)
  "Writes the play log, along with -- o número de nós analisados, o número de
cortes efetuados (de cada tipo),    o tempo gasto em cada jogada e o tabuleiro
atual."

    (with-open-file (file (statistics-file-path) :direction :output :if-exists
:append :if-does-not-exist :create)
           (progn
             (terpri)
                (format file "~%~t------------:PLAY MADE------------")
             (terpri)
                (format file "~%~t----:  Playing Time: ~a Miliseconds" play-time)
             (terpri)
                (format file "~%~t----:  Points Player-1: ~a" points-1)
             (terpri)
                (format file "~%~t----:  Points Player-2: ~a" points-2)
            (terpri)
                (format file "~%~t----:  Alpha Cuts: ~a" alpha-cuts)
            (terpri)
                (format file "~%~t----:  Beta Cuts: ~a" beta-cuts)
            (terpri)
                (format file "~%~t----:  Number of Analized Nodes: ~a" num-
analized-nodes)

             (terpri)
             (terpri)
                (format file "~%~t----:  Current Board:")
             (terpri)
                (print-board board file)
             )
    )
  )
```

Limitações

Profundidade

Em termos de limitações, o ficheiro de algoritmo poderia estar mais modular, isto é, mais idependente do domínio do jogo do cavalo, com menos código repetido, e melhorado em termos de refatoração. O algoritmo

NEGAMAX é limitado de forma a que existe um problema com a noção do backup da "árvore" gerada. Quando a profundidade máxima inserida pelo utilizador é maior do que 1, existe a possibilidade de um movimento ilegal de um cavalo seja executado, pois em vez de se devolver o valor da jogada correspondente ao nível 1 da árvore, devolve valores correspondentes a níveis maiores.

Valor Heurístico

Os valores heurísticos de cada nó são calculados assim que são "construídos" pela sua função construtora. Contudo, devido a inicialmente termos desenvolvido o algoritmo e a forma como o jogo executa jogadas contando com que o valor a avaliar é o valor de pontos da casa atual, o que fez com que, devido a restrições de tempo, não pudéssemos alterar a funcionalidade do programa para que verificasse o valor heurístico de cada nó, que foi aplicado ao programa já pouco tempo antes da entrega deste projeto.

Qualidade do Código

Devido a tentar-mos resolver os problemas descritos anteriormente, a qualidade do código no geral acavou por ficar afetada, verificando-se a presença de código repetido e "pouco arrumado", assim como a presença de funções que não são utilizadas.

Tempo de Jogada

Após uma análise das estatísticas, foi possível observar que existe um problema com a verificação do tempo máximo de jogada, que não consigos resolver a tempo. Com isto, parece ser possível uma jogada demorar mais do que o tempo máximo.

Simétrico

A remoção do valorr simétrico é aplicada apenas a partir da segunda jogada de cada jogador. Este problema deveu-se a que a regra do simétrico só foi implementada muito pouco antes da entrega deste projeto, por isso este problema não foi resolvido

Demonstação de Limitações

Abaixo emcontra-se dois movimentos seguidos do jogador -1, com uma profundidade máxima atribuída ao NEGAMAX de 1, em que o útimo é ilegal.

Profundidade 1

```
Computer -1 move: I4

(75 73 42 32 5 49 25 30 37 81)
(NIL 10 2 8 26 7 19 40 20 58)
(67 16 NIL NIL -2 70 NIL 54 43 48)
(13 NIL 82 22 NIL NIL 38 0 39 79)
```

```
(33 59 41 68 NIL NIL 65 31 11 52)
(NIL 45 NIL 24 NIL 46 62 NIL 35 27)
(64 12 NIL 56 17 NIL 15 57 84 NIL)
(66 34 47 NIL NIL 23 72 NIL 76 63)
(14 3 18 -1 1 NIL NIL 6 NIL NIL)
(4 74 36 NIL NIL NIL NIL 21 80 28)
```

```
Computer -1 move: J2

(75 73 42 32 5 49 25 30 37 81)
(NIL 10 2 8 26 7 19 40 20 58)
(67 16 NIL NIL NIL 70 NIL 54 43 48)
(13 NIL -2 22 NIL NIL 38 0 39 79)
(33 59 41 68 NIL NIL 65 31 11 52)
(NIL 45 NIL 24 NIL 46 62 NIL 35 27)
(64 12 NIL 56 17 NIL 15 57 84 NIL)
(66 34 47 NIL NIL 23 72 NIL 76 63)
(14 3 18 NIL 1 NIL NIL 6 NIL NIL)
(4 -1 36 NIL NIL NIL NIL 21 80 28)
```

Profundidade 3

Abaixo emcontra-se dois movimentos seguidos do jogador -1, com uma profundidade máxima atribuída ao NEGAMAX de 3, em que o útimo é ilegal.

```
Computer -1 move: F8

(85 92 2 NIL NIL 0 80 NIL 33 NIL)
(78 11 NIL 29 NIL 14 NIL NIL 18 12)
(61 67 NIL 65 88 37 77 NIL 39 36)
(25 48 NIL NIL 55 NIL 17 NIL NIL NIL)
(27 4 3 NIL NIL 22 NIL 54 40 32)
(47 34 97 5 NIL 10 87 -1 -2 59)
(7 96 NIL 30 31 23 43 26 NIL 46)
(58 13 NIL 38 24 15 51 NIL 21 NIL)
(6 75 35 19 63 NIL 52 60 NIL NIL)
(1 57 45 62 8 53 NIL 50 72 16)
```

```
Computer -1 move: F7

(85 92 2 NIL NIL 0 80 NIL 33 NIL)
```

```
(78 11 NIL 29 NIL 14 NIL NIL 18 12)
(61 67 NIL 65 88 37 77 NIL 39 36)
(25 48 NIL NIL 55 NIL 17 NIL NIL NIL)
(27 4 3 NIL NIL 22 NIL 54 40 32)
(47 34 97 5 NIL 10 -1 NIL NIL 59)
(7 96 NIL 30 31 23 43 26 NIL 46)
(58 13 NIL 38 24 15 51 NIL 21 NIL)
(6 75 35 19 63 NIL 52 60 NIL NIL)
(1 57 45 62 8 -2 NIL 50 72 16)
```

Análise Estatística

Abaixo estão exemplos das estatísticas obtídas após alguns movimentos do computador contra o humano. Com uma profundiade de 1, não existem cortas alpha-beta mas é possível visulizar o número de nós que são analisados, sendo o último diferente dos restantes

```
------------:PLAY MADE------------
 ----:  Playing Time: 2 Miliseconds
 ----:  Points Player-1: 170
 ----:  Points Player-2: 266
 ----:  Alpha Cuts: 0
 ----:  Beta Cuts: 0
 ----:  Number of Analized Nodes: 7
 ----:  Current Board:
   (38 NIL 80 53 43 56 9 55 34 32)
   (60 46 13 65 75 30 12 57 21 89)
   (44 3 NIL 87 70 63 25 20 26 7)
   (92 4 54 82 -1 29 23 97 76 94)
   (96 71 93 37 51 15 2 52 91 16)
   (64 58 28 5 69 79 39 -2 85 95)
   (24 31 8 50 41 86 90 40 48 98)
   (88 18 67 19 14 0 NIL 33 1 72)
   (81 62 74 45 42 59 35 47 22 27)
   (10 6 78 77 49 17 61 NIL 11 73)
 ------------:PLAY MADE------------
 ----:  Playing Time: 3 Miliseconds
 ----:  Points Player-1: 175
 ----:  Points Player-2: 364
 ----:  Alpha Cuts: 0
 ----:  Beta Cuts: 0
 ----:  Number of Analized Nodes: 7
 ----:  Current Board:
   (38 NIL 80 53 43 56 9 55 34 32)
   (60 46 13 65 75 30 12 57 21 89)
   (44 3 NIL 87 70 63 25 20 26 7)
   (92 4 54 82 NIL 29 23 97 76 94)
   (96 71 93 37 51 15 2 52 91 16)
```

```
  (64 58 28 -1 69 79 39 NIL 85 95)
  (24 31 8 50 41 86 90 40 48 -2)
  (88 18 67 19 14 0 NIL 33 1 72)
  (81 62 74 45 42 59 35 47 22 27)
  (10 6 78 77 49 17 61 NIL 11 73)
------------:PLAY MADE------------
----:  Playing Time: 4 Miliseconds
----:  Points Player-1: 261
----:  Points Player-2: 455
----:  Alpha Cuts: 0
----:  Beta Cuts: 0
----:  Number of Analized Nodes: 3
----:  Current Board:
  (38 NIL 80 53 43 56 9 55 34 32)
  (60 46 13 65 75 30 12 57 21 89)
  (44 3 NIL 87 70 63 25 20 26 7)
  (92 4 54 82 NIL 29 23 97 76 94)
  (96 71 93 37 51 15 2 52 -2 16)
  (64 58 28 NIL 69 79 39 NIL 85 95)
  (24 31 8 50 41 -1 90 40 48 NIL)
  (88 18 67 19 14 0 NIL 33 1 72)
  (81 62 74 45 42 59 35 47 22 27)
  (10 6 78 77 49 17 61 NIL 11 73)
```

Para um profundidade de 4, embora com movimentos ilegais, é possível visualizar a o número de cortes alpha-beta realizados e um número de nós analisados muito maior.

```
------------:PLAY MADE------------
 ----:  Playing Time: 148 Miliseconds
 ----:  Points Player-1: 55
 ----:  Points Player-2: 153
 ----:  Alpha Cuts: 16
 ----:  Beta Cuts: 43
 ----:  Number of Analized Nodes: 292
 ----:  Current Board:
   (86 81 17 34 87 23 20 39 91 -1)
   (22 33 42 74 60 96 0 61 28 44)
   (36 41 93 69 10 31 1 56 7 27)
   (73 71 50 8 -2 29 64 9 90 62)
   (52 14 95 24 79 21 98 3 19 77)
   (13 83 46 40 66 85 5 32 48 68)
   (51 47 15 45 4 70 72 84 59 NIL)
   (25 11 38 94 12 76 75 92 78 53)
   (58 26 18 2 82 88 97 63 37 89)
   (43 30 67 57 49 6 16 80 35 65)
 ------------:PLAY MADE------------
 ----:  Playing Time: 167 Miliseconds
 ----:  Points Player-1: 62
 ----:  Points Player-2: 222
```

```
----:  Alpha Cuts: 27
----:  Beta Cuts: 72
----:  Number of Analized Nodes: 371
----:  Current Board:
  (86 81 17 34 87 23 20 39 91 NIL)
  (22 33 42 74 60 96 0 61 28 44)
  (36 41 93 -2 10 31 1 56 -1 27)
  (73 71 50 8 NIL 29 64 9 90 62)
  (52 14 95 24 79 21 98 3 19 77)
  (13 83 46 40 66 85 5 32 48 68)
  (51 47 15 45 4 70 72 84 59 NIL)
  (25 11 38 94 12 76 75 92 78 53)
  (58 26 18 2 82 88 97 63 37 89)
  (43 30 67 57 49 6 16 80 35 65)
```