Dots and Boxes: Manual Técnico

Inteligência Artificial - Escola Superior de Tecnologia de Setúbal

André Dias - 201901690

João Caetano - 201901961

2022/2023

Prof. Joaquim Filipe Eng. Filipe Mariano

1. Introdução

Este documento representa uma mais valia para técnicos não participantes do desenvolvimento do projeto para melhor entender as especificidades técnicas do programa. O projeto *Dots and Boxes* visa proporcionar uma versão simplifica do mesmo jogo, no intuito de se estudar o campo da teoria de jogos. Todo o código foi implementado recorrendo 100% à linguagem ***Common-Lisp***.

2. Arquitetura do sistema

O projeto desenvolvido encontra-se separado em 3 sistemas, cada um deles implementado em um ficheiro próprio.

2.1 Jogo.lisp

O ficheiro jogo.lisp é o sistema de entrada para o utilizador. Por isso, estará apenas encarregue de tratar do carregamento dos outros 2 sistemas e de iniciação do programa, bem como de executar o jogo. Estas ações são feitas por meio da função

```
(defun start ()
"Program start function."
    (load (compile-file "puzzle.lisp"))
    (load (compile-file "algoritmo.lisp"))
    (load (compile-file "jogo.lisp"))
    (main-menu)
)
```

Após a execução da função (main-menu), o flow de UI/UX começa, de forma "sequenciada" (apenas o flow é considerado sequenciado, as funções são chamadas recursivamente até haver paragem no programa). Ao utilizador será apresentado a escolha dos dois modos de jogo disponíveis, por meio da função.

```
(defun game-mode-menu ()
"Displays and handles game mode selection logic."
    (reset-vars nil)
    (format t "Pick the desired game mode: ~%")
    (format t "1 - You Vs AI~%")
```

```
    (format t "2 - Clash of bots (AI vs AI)~%~%")
    (format t ">")
    (let ((inp (read)))
        (cond ((eql inp 'quit) (cl-user::quit))
              ((or (not (numberp inp)) (> inp 2) (< inp 1)) (error-handle 'game-
mode-menu "~%Error: Please input a valid option~%~%"))
              ((= inp 1) (human-bot-game (list (starting-board) 0 0)))
              ((= inp 2) (bot-bot-game (list (starting-board) 0 0)))
        )
    )
)
```

Após a escolha do jogo, será invocada a função para gerir o tipo de jogo escolhido.

2.2 Puzzle.lisp

Este sistema está apenas encarregue da manipulação do tabuleiro do jogo e dos nós gerados. Cada nó tem a seguinte estrutura:

```
'(estado nó-pai profundidade valor-heuristico)
```

O estado é representado da seguinta forma:

```
'(tabuleiro pontuação-jogador-1 pontuação-jogador-2)
```

E o tabuleiro é representado da seguinte maneira:

```
'(
    (
        (0 0 0 0 0 0) "arcos horizontas/linhas"
        (0 0 0 0 0 0)
        (0 0 0 0 0 0)
        (0 0 0 0 0 0)
        (0 0 0 0 0 0)
        (0 0 0 0 0 0)
    )
    (
        (0 0 0 0 0) "arcos verticais/colunas"
        (0 0 0 0 0)
        (0 0 0 0 0)
        (0 0 0 0 0)
        (0 0 0 0 0)
        (0 0 0 0 0)
        (0 0 0 0 0)
    )
)
```

Os números representam os arcos, que são os espaços entre dois pontos adjacentes. Nas linhas, os arcos são horizontais. Nas colunas, os arcos são verticais.

Dentro deste sistema, poderá fazer-se alterações ao tabuleiro ao inserir arcos numa posição (linha x coluna y) para um dado jogador (1 ou 2). Na lista das linhas, (linha x) representa a sublista x, e a (coluna y) representa o elemento y da sublista x. Na lista de colunas, o comportamento é feito ao contrário: elemento x da sublista y. Estas operações são utilizadas para se gerarem as próximas jogadas possíveis para um dado jogador, dado o estado de um tabuleiro, ou para efetuar uma jogada manual.

2.3 Algoritmo.lisp

Por último, o sistema de procura implementa o algoritmo negamax com cortes alfa-beta. Este sistema apenas gere a procura da melhor jogada dado um nó como raíz, até uma profundidade máxima para um dado jogador maximizante.

```
(defun negamax (node depth player a b color &optional (max-player 2))
"Implementation of the negamax algorithm with alpha beta cuts. This is the fail-
soft version. Root node is NODE, with max depth and player number for successor
generation.
Color represents the signal switching of a terminal node's value. Max-player is
the player number that the algorithm is trying to maximize, 2 by default."
    (if (or (terminal-p node) (= depth 0)) ;Se o nó for terminal ou esteja em
profundidade limite:
        (* color (eval-node node max-player)) ;Retorna-se valor * sinal para este
nó.
        (let ((successors (qsort-nodes (generate-successors node (all-actions-
list) player))) ;Sorting dos nós.
        (value most-negative-fixnum))
            (dolist (child successors)
                (setf value (max value (- (negamax child (1- depth) (switch-player
player) (- b) (- a) (- color) max-player))))
                (progn (when (and (null (get-parent-node node)) (< (second
*optimal-play*) value)) (setf *optimal-play* (list (get-node-state child)
value)))) ; Guarda-se a próxima melhor jogada se o valor for melhor.
                (setf a (max a value))
                (setf *explored* (1+ *explored*))
                (when (>= a b) (progn (setf *cuts* (1+ *cuts*))(return))) ;Corte
            )
            value
        )
    )
)
```

A versão deste algoritmo negamax com cortes alfa-beta é a versão fail-soft. Não houve um fator decisivo para a escolha.

Como limitação, esta aproximação ao desenvolvimento do algoritmo não é ideal, pois utiliza macros de loops (dolist) com mudança de variáveis (setf). No entanto, dado a situação académica em termos de carga horária do grupo, optou-se por poupar tempo ao evitar encontrar uma solução puramente recursiva que aderisse a todas as boas prática de Common Lisp.

3. Entidades e sua implementação

Para além dos nós (já anteriormente específicado), tecnicamente a variável * **optimal-play** * poderá ser considerada uma estrutura por si só, que consiste numa lista com:

- Estado do problema: Tabuleiro da próxima jogada e pontuação dos dois jogadores;
- Valor da função de avaliação para a jogada encontrada.

```
(list state value)
```

4. Algoritmos e implementação

Para o algoritmo da teoria de jogos, escolheu-se o Negamax por apresentar a estrutura mais simples face ao Minimax.

5. Análise estatística

Para analisar o comportamento do algoritmo negamax implementado, extraíu-se do ficheiro log.dat parte dos resultados de um jogo entre um humano e o *bot*.

Jogada	Nós cortados	Nós explorados
1	69	2185
2	66	296
3	64	1243
4	62	252
5	61	184
6	58	1442
7	57	172
8	55	1255
9	52	1515
10	49	338

Por motivos de extensibilidade, apenas estão representadas 10 jogadas do computador, mas são suficientes para mostrar a tendência decrescente dos nós cortados. Isto acontece porque, à medida que o tabuleiro enche, cada estado tem menos jogadas possíveis, os nós têm menos filhos. Também é de notar os picos nos nós explorados em certas jogadas. Isto deve-se ao facto de que, sempre que o jogador humano mete em risco a posição do computador, mais casos são analisados (poderá ser um *side effect* da profundidade reduzida).

6. Limitações técnicas

No contexto da UC de IA, o requisito de tempo limite não foi implementado por constrangimento temporal. Isto leva a que limites de profundidade muito altos (acima de 5) façam com que a procura negamax se atrase consideravelmente. Posto isto, limitou-se a usar uma profundidade (talvez demasiado baixa) de 2.