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 estudarem diversos algoritmos de procura em árvores. 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 Projeto.lisp

O ficheiro Projeto 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 executar o algoritmo com o problema escolhido. Estas ações são feitas por meio da função

```
(defun load-lisp-files ()
"Loads and compiles all of the program's files."
    (load (compile-file "puzzle.lisp"))
    (load (compile-file "procura.lisp"))
    (load (compile-file "projeto.lisp"))
)
```

E da função:

```
(defun start ()
"Starts the program."
    (load-lisp-files)
    (main-menu)
)
```

Após a execução da função (main-menu), o flow de UI/UX começa, de forma sequênciada, ao passar por parametro do próximo menu os dados escolhidos no menu corrente. Por exemplo:

```
(defun algorithm-menu (problem)
"Algorithm menu function. Given PROBLEM, user can select which algorithm to search
with."
    (format t "Choose the searching algorithm:~%~%1.BFS~%2.DFS~%3.A*~%")
    (let ((inp (read)))
        (if (or (not (numberp inp)) (> inp 3) (< inp 1))
            (algorithm-menu problem)
            (ecase inp
                (1 (run-search problem 'bfs))
                (2 (depth-menu problem 'dfs))
                (3 (heuristic-menu problem 'a*))
            )
        )
    )
)
```

Onde (run-search), (depth-menu) e (heuristic-menu) recebem o problema escolhido no menu anterior. Este sistema interage diretamente apenas com o sistema de procura, ao qual envia call da função de procura a executar e recebe o seu resultado.

2.2 Puzzle.lisp

Este sistema está apenas encarregue da manipulação do tabuleiro do jogo, que é representado da seguinte maneira:

```
(
    ((0 0 0)(0 0 0)(0 0 1)) ;Representação das linhas
    ((1 0 0)(0 0 0)(0 1 0)) ;Representação das colunas
)
```

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). 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 no sistema Procura, para se gerarem as próximas jogadas possíveis, dado o estado de um tabuleiro.

2.3 Procura.lisp

Por último, o sistema de procura implementa os algoritmos de procura, bem como as suas funções auxiliares para a criação de tipos de dados. Este sistema recebe dados do utilizador, tratados pelo sistema Projeto, que despolotam a geração da árvore de procura, dado um determinado problema e operadores definidos do sistema Puzzle.

3. Entidades e sua implementação

A principal entidade implementada no projeto, foi a representação dos nós da árvore: os objetos Node. Um Node é representado por uma lista que, dentro dela, contem a seguinte informação:

```
('state 'parent-node 'node-level 'h)
```

Onde:

- state: Estado do nó (o tabuleiro, neste caso)
- parent-node: Nó que gerou este nó
- node-level: Profundidade na árvore a que o nó se encontra
- h: Valor heurístico determinado por uma dada função heurística

No final da procura, é apresentada uma outra entidade: solution. Este objeto contém a informação toda necessária para dar ao utilizador a solução de um problema num formato mais amigável.

```
('objective-node 'total-nodes 'total-expanded-nodes 'penetrance 'effective-
branching-factor 'runtime)
```

Onde:

- objective-node: Nó solução
- total-nodes: Número total de nós gerados pelo algoritmo
- total-expanded-nodes: Número de nós explorados pelo algoritmo
- penetrance: Penetrancia (relação entre profundidade do nó objetivo com o número total de nós)
- effective-branching-factor: Fator de ramificação média (Quantos nós são gerados em média por nível)
- runtime: Tempo medido pelo ambiente entre início e fim do algoritmo (em microsegundos)

4. Algoritmos e implementação

No sistema de procura, os seguintes algoritmos foram implementados:

- BFS: Breadth-first-search
- DFS: Depth-first-search
- A*: A-star

Para cada um dos algoritmos é possível apresentar as métricas de desempenho de tempo, penetrância e ramificação média, para que se possa comparar a eficiência entre cada um deles.

4.1 Amostras de desempenho em cada problema

Estes resultados foram calculados através da execução do programa na implementação de GNU CLISP 2.49 utilizando a terminal nativa do computador onde foi executado, no ambiente Windows 11. O processador utilizado foi o Intel Core i5 11400F @ 2.6GHz.

Problema 1

Nº Nós gerados	Nº Nós expandidos	Algoritmo	Tempo	Penetrância	Fator Ramificação Média
91	6	BFS	0.0ms	0.022	2.434
91	66	DFS	0.01ms	0.022	2.043
22	2	A*	0.01ms	0.091	2.039
22	2	A* (heuristica nova)	0.01ms	0.091	2.039

Problema 2

Nº Nós gerados	Nº Nós expandidos	Algoritmo	Tempo	Penetrância	Fator Ramificação Média
15	1	BFS	0.0ms	0.067	8.027
15	1	DFS	0.01ms	0.067	8.027
15	1	A*	0.01ms	0.067	8.027
15	1	A* (heuristica nova)	0.01ms	0.067	8.027

Problema 3

A partir deste problema torna-se impossível e/ou pouco viável medir o desempenho do BFS e DFS.

Nº Nós gerados	Nº Nós expandidos	Algoritmo	Tempo	Penetrância	Fator Ramificação Média
134	10	A*	0.05ms	0.075	10.03
134	10	A* (heuristica nova)	0.06ms	0.075	10.03

Problema 4

Nº Nós gerados	Nº Nós expandidos	Algoritmo	Tempo	Penetrância	Fator Ramificação Média
1146	37	A*	0.55ms	0.032	37.034
971	27	A* (heuristica nova)	0.6ms	0.027	27.029

Problema 5

Nº Nós gerados	Nº Nós expandidos	Algoritmo	Tempo	Penetrância	Fator Ramificação Média

Nº Nós gerados	Nº Nós expandidos	Algoritmo	Tempo	Penetrância	Fator Ramificação Média
535	16	A*	0.31ms	0.03	16.032
535	16	A* (heuristica nova)	0.49ms	0.03	16.032

Problema 6

Nº Nós gerados	Nº Nós expandidos	Algoritmo	Tempo	Penetrância	Fator Ramificação Média
5858	93	A*	9.70ms	0.016	93.044
5719	86	A* (heuristica nova)	11.63ms	0.015	86.043

5. Descrição das opções tomadas

Todas as opções de implementação foram decididas à volta das convenções de linguagem ***Common Lisp***, não havendo em lugar algum a utilização de funções destrutivas ou que possam produzir side-effects, nem definições de variáveis globais.

Na parte de procura, implementou-se um algoritmo de ordenação para as listas de nós. No final escolheu-se o quick sort pela sua eficiência que poderá não ter sido a melhor ideia, pois o seu comportamento pode não ser estável, alterando a complexidade de tempo de execução. Possivelmente, um merge sort poderá ter sido mais consistente.

6. Limitações técnicas

No contexto da UC de IA, o único requisito não implementado foi o algoritmo de procura bónus. No contexto do programa, um refactoring na interação de utilizador poderá ser uma mais valia. Desacoplar as funções de menu para que se possa voltar atrás. Mais ainda, a função de gerar os sucessores de um nó está fortemente acoplada ao problema.