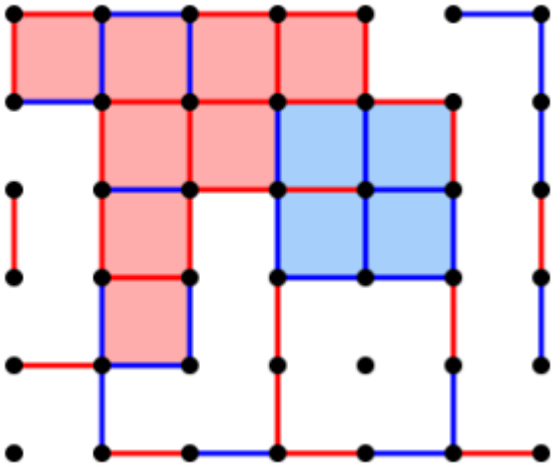
Manual Técnico

Jogo 'Dots and Boxes'



Curso:

Licenciatura em Engenharia Informática

Unidade Curricular:

Inteligência Artificial

Docente de Laboratório:

Filipe Mariano

Trabalho elaborado por:

Diogo Filipe Brália Letras, Nº 202002529

Pedro Manuel Fialho Cunha, Nº 20200757

Índice

Introdução

No âmbito da unidade curriclar de Inteligência Artificial, os docentes propuseram o desenvolvimento de um projeto em LISP intitulado "Dots and Boxes". O objetivo deste projeto é aplicar os conhecimentos adquiridos durante as aulas na criação de um jogo do tipo puzzle.

O puzzle consiste em fechar um determinado número de caixas a partir de uma configuração inicial do tabuleiro. Para isso, será necessário desenhar arcos entre dois pontos adjacentes na horizontal ou na vertical. Quando quatro arcos são desenhados entre pontos adjacentes, uma caixa é fechada.

Ambiente de desenvolvimento

Abaixo, podem ser visualizadas as ferramentas utilizadas durante o desenvolvimento do projeto.

Ferramentas	Propósito
LispWorks	Utilizada no desenvolvimento
Visual Studio Code	Utilizada na escrita do relatório
Github	Ferramenta para gerir as versões de controlo.

Requisitos de hardware das ferramentas utilizadas

Segue-se os requisitos necessários para poder correr as ferramentas sem constrangimentos.

Nota: os seguintes requisitos supõem que a máquina está a correr o sistema operativo Windows (64 bits).

Ferramentas	Processador	Espaço no disco
LispWorks	AMD64, Intel64	170MB
Visual Studio Code	1.6Ghz ou mais rápido	500MB
Github	-	-

Estrutura do projeto

O projeto contém três ficheiros principais e um ficheiro auxiliar:

- Projeto.lisp: este ficheiro é responsável pela interação com o utilizador, mostrando menus no ecrã com as opções disponíveis.
- Puzzle.lisp: este ficheiro é responsável pela gestão das regras do jogo, incluindo entre outros: operadores, validações e outras funções úteis para o desenvolvimento do projeto.
- Procura.lisp: este ficheiro contém tudo o que esteja relacionado a parte algorítmica. Inclui funções úteis para monitorar o desempenho dos algoritmos tais como fator de ramificação e penetrância. É responsável por executar a busca de uma solução para cada problema.
- Problemas.dat: este ficheiro contém os tabuleiros que serão resolvidos pelo programa. Cada tabuleiro é composto por uma lista de arcos verticais e horizontais.

Algoritmos

Foram propostos 3 algoritmos para serem implementados e também outros 3 algoritmos opcionais.

De seguida, serão detalhados os algoritmos que foram implementados no projeto.

Breadth First Search

O algoritmo BFS (em inglês, "Breadth-First Search") é um algoritmo de busca não-informado (não utiliza heurística) que é amplamente utilizado em problemas que envolvem o caminho mais curto entre dois pontos. Funciona explorando todos os nós adjacentes ao nó atual antes de passar para os próximos níveis da árvore. Dessa forma, o algoritmo garante que o caminho mais curto entre o nó inicial e qualquer outro nó será encontrado. O algoritmo BFS pode ser implementado de forma iterativa ou recursiva.

Depth First Search

O algoritmo DFS (em inglês, "Depth-First Search") é um algoritmo de busca não-informado (não utiliza heurística) que é amplamente utilizado em problemas que envolvem a busca em profundidade. Funciona explorando o nó atual e, em seguida, explorando recursivamente todos os nós adjacentes a ele antes de passar para o próximo nó na árvore. Dessa forma, o algoritmo DFS garante que todos eles serão visitados antes de serem explorados. O algoritmo DFS também pode ser implementado de forma iterativa ou recursiva.

A* Search Algorithm

O A*("A-star") Search Algorithm é um algoritmo de busca informado usado para encontrar o caminho mais curto numa árvore. O algoritmo combina os pontos fortes da busca em largura e da busca de custo uniforme e é considerado um dos algoritmos de pesquisa mais eficazes. O algoritmo A* funciona mantendo uma lista de nós que foram visitados e uma lista de nós que ainda serão visitados. O algoritmo começa no nó inicial e depois expande o nó mais promissor, com base numa função heurística que estima o custo para atingir o objetivo a partir desse nó. O algoritmo então continua a explorar os nós, sempre escolhendo o nó com o menor custo total estimado (que é a soma do custo para atingir esse nó desde o início e o custo estimado para atingir o objetivo a partir desse nó).

Funções

De seguida, apresenta-se as principais funções do programa, juntamente com a sua descrição.

Puzzle.lisp

Função	Descrição
(create-node)	Cria uma lista que representa um nó. Um nó é composto por: - estado, que neste caso é o tabuleiro, - profundidade a que nó se encontra, - heurística do nó - nó pai.
(get-node-state)	Retorna o estado do nó.
(get-node-depth)	Retorna a profundidade do nó.
(get-node-heuristic)	Retorna a heurística do nó.

Função	Descrição
(get-node-parent)	Retorna o pai do nó.
(get-node-cost)	Retorna o valor do nó, ou seja a soma do valor da profundidade (g) com o valor da heurística (h).
(get-horizontal-arcs)	Retorna a lista de arcos horizontais no tabuleiro.
(get-vertical-arcs)	Retorna a lista de arcos verticais no tabuleiro.
(insert-arc-in-positional-final)	Insere um arco representado pelo valor 1 numa lista que representa o conjunto de arcos no tabuleiro.
(verify-add-arc)	Verifica se qual o valor de uma determinada posição numa lista de arcos. Se o valor for 0, retorna 1 e se o valor for 1 retorna 0.
(insert-horizontal-arc)	Insere um arco horizontal, representado pelo valor 1, no tabuleiro.
(insert-vertical-arc)	Insere um arco vertical, representado pelo valor 1, no tabuleiro.
(operators)	Cria uma lista com todos os operadores do problema.
(num-closed-boxes)	Retorna o número de caixas fechadas no tabuleiro.
(solutionp)	Se o valor de caixas a fechar for igual ao número de caixas fechadas, retorna 1. Caso contrário retorna 0.
(path-solution)	Retorna o caminho da solução, ou seja, todos os estados desde o estado inicial até ao estado final.
(original-heuristic)	Função heurística proposta pelos docentes.
(alternative-heuristic)	Função heurística proposta pelos alunos.

Procura.lisp

Função	Descrição
(successors)	Retorna a lista de sucessores de um nó.

Função	Descrição
(generic-search)	Implementa um algoritmo que efetua uma procura genérica num espaço de estados. Pode ser utilizado em conjunto com o algoritmo de largura (bfs), o algoritmo de profundidade (dfs) ou o algoritmo a-asterisco (a*).
(bfs)	Executa uma procura em largura. Coloca os nós sucessores no final da lista de abertos (nós não visitados).
(dfs)	Executa uma procura em profundidade. Coloca os nós sucessores no início da lista de abertos (nós não visitados).
(a-asterisc)	Executa uma procura com o algoritmo a*. Coloca os nós sucessores no final da lista de abertos (nós não visitados) e efetua uma ordenação dos nós pelo seus custos.
(existp)	Verifica se um determinado nó já existe numa lista de nós. Utilizado no algoritmo dfs.
(exist-solution)	Verifica se uma solução existe numa lista de nós sucessores. Utilizado no algoritmo dfs.
(penetrance)	Calcula o valor da penetrância.
(branching-factor)	Calcula o valor médio do fator de ramificação.

Projeto.lisp

Função	Descrição
(start)	Inicializa o programa.
(insert-directory)	Pede ao utilizador para inserir a diretoria onde estão localizados os ficheiros do programa.
(compile-load-files)	Efetua a compilação e carrega os ficheiros necessários para o funcionamento do programa.
(main-menu)	Desenha o menu principal da aplicação.
(start-search)	Pede ao utilizador a informação necessária para iniciar uma procura no espaço de estados, nomeadamente: o estado inicial (tabuleiro), algoritmo de procura e possivelmente também a profundidade máxima.
(read-boards)	Mostra todos os tabuleiros possíveis do jogo. Recebe a escolha do tabuleiro do utilizador e verifica se a escolha é válida.
(read-num-boxes-to-close)	Recebe o número de caixas a fechar no tabuleiro.
(read-algorithm)	Recebe o nome do algoritmo que irá ser utilizado na procura.

Função	Descrição
(read-heuristic)	Recebe o nome da função heurística a ser utilizada no algoritmo a*.
(read-depth)	Recebe a profundidade máxima. Utilizado no algoritmo bfs.
(puzzle-rules)	Desenha as regras do puzzle 'dots and boxes'.
(write-stats)	Escreve as estatísticas de uma procura no ficheiro estatisticas.dat e também na consola.
(exception-stats)	Escreve no ficheiro estatísticas.dat que não foi possível calcular as estatísticas de uma procura.

Análise dos resultados

De seguida apresenta-se os resultados da análise aos algoritmos.

Problema A -- 3 caixas a fechar

	BFS	DFS	A* HeuristicaOG	A* HeuristicaALT
Tempo decorrido	0	0	0	0
Profundidade	2	6	2	2
Nós gerados	1129	82	313	298
Nós expandidos	81	6	22	21
Penetrância	0.0017714792	0.07317073	0.0063897763	0.0067114094
Fator de ramificação	8.531752	1.0284217	4.2116528	4.097966

Problema B -- 7 caixas a fechar

	BFS	DFS	A* HeuristicaOG	A* HeuristicaALT
Tempo decorrido	0	0	0	0
Profundidade	1	1	1	1
Nós gerados	198	16	198	198
Nós expandidos	15	1	15	15
Penetrância	0.005050505	0.0625	0.005050505	0.005050505
Fator de ramificação	15.125589	1.0852652	15.125589	15.125589

Problema C -- 10 caixas a fechar

	BFS	DFS	A* HeuristicaOG	A* HeuristicaALT

	BFS	DFS	A* HeuristicaOG	A* HeuristicaALT
Tempo decorrido	-	0	-	-
Profundidade	-	14	-	-
Nós gerados	-	162	-	-
Nós expandidos	-	14	-	-
Penetrância	-	0.086419754	-	-
Fator de ramificação	-	1.0284217	-	-

Problema D -- 10 caixas a fechar

	BFS	DFS	A* HeuristicaOG	A* HeuristicaALT
Tempo decorrido	-	0	-	-
Profundidade	-	34	-	-
Nós gerados	-	800	-	-
Nós expandidos	-	34	-	-
Penetrância	-	0.0425	-	-
Fator de ramificação	-	1.0284217	-	-

Problema E -- 20 caixas a fechar

	BFS	DFS	A* HeuristicaOG	A* HeuristicaALT
Tempo decorrido	-	0	-	-
Profundidade	-	30	-	-
Nós gerados	-	796	-	-
Nós expandidos	-	30	-	-
Penetrância	-	0.03768844	-	-
Fator de ramificação	-	1.0284217	-	-

Problema F -- 35 caixas a fechar

	BFS	DFS	A* HeuristicaOG	A* HeuristicaALT
Tempo decorrido	-	0	-	-
Profundidade	-	95	-	-
Nós gerados	-	5891	-	-
Nós expandidos	-	95	-	-

	BFS	DFS	A* HeuristicaOG	A* HeuristicaALT
Penetrância	-	0.016126294	-	-
Fator de ramificação	-	1.0284217	-	-

Conclui-se que o algoritmo BFS tem um desempenho razoável quando o número de caixas a fechar é próximo do número de caixas pré-preenchidas. Isto acontece porque o BFS expande primeiro os nós mais próximos do nó inicial.

O algoritmo DFS é claramente melhor quando o número de caixas a fechar é distante longe do número de caixas pré-preenchidas. Isto acontece porque o DFS pode explorar mais profundamente a árvore antes de retroceder ("backtracking").

Algoritmo A*

Foram testadas duas heurísticas:

Heurística Original

h0(x) = o(x) − c(x)

em que:

- o(x) é o objetivo para esse tabuleiro: o número de caixas a fechar no tabuleiro x,
- c(x) é o número de caixas já fechadas no tabuleiro x.

Heurística Alternativa (criada pelos alunos)

*h1(x) = h0(x) * (board-size) + 1*

em que:

- h0(x) é a heurística original,
- (board-size) é o tamanho do tabuleiro

Este algoritmo no geral é melhor do que o algoritmo BFS, pois é um algoritmo informado, ou seja, utiliza uma heurística. No entanto, é pior do que o algoritmo DFS, pois pode gerar nós repetidos, o que prejudica o seu desempenho. A heurística alternativa conseguiu ser ligeiramente melhor que a heurística original.

Nota: A partir do problema C em diante, foi não possível usar o algoritmo bfs e o algoritmo a*, pois a versão gratuita do LispWorks tem uma limitação de memória, o que fez com que ocorresse uma mensagem de erro cada vez que se tentava correr esses algoritmos.

Limitações do projeto

A principal limitação da aplicação é o facto de, em tabuleiros onde existam poucas caixas pré-preenchidas, o algoritmo BFS e o algoritmo A* não têm possibilidade de terminar a execução por falta de memória. Isto, deve-se à limitação de memória imposta pelo LispWorks, o IDE utilizado no projeto.

Todos os requisitos obrigatórios do projeto foram implementados.

Não foram implementados os algoritmos bónus.

Conclusão

A realização deste projeto foi algo desafiante, pois a aplicação foi desenvolvida na linguagem Lisp, com um paradigma funcional. Esta abordagem à programação difere em relação ao que os membros do grupo estavam habituados até agora.

O projeto serviu também para solidificar alguns conceitos que foram abordados em outras unidades curriculares anteriores e que foram novamente revisitados nesta unidade curricular de Inteligência Artificial, nomeadamente os algoritmos em largura e profundidade e a recursividade.

Portanto, sem dúvidas que a experiência de desenvolvimento deste projeto, deverá ser útil no futuro académico e profissional dos membros do grupo.