Manual Técnico

Logotipo da Instituto Politécnico de Setúbal - Escola Superior de Tecnologia de Setúbal

Puzzle dos Pontos e das Caixas

**Docentes da disciplina**

Joaquim Filipe

Cédric Grueau

**Um trabalho realizado por**

Artur Esteves 140221076 Engenharia Informática

Daniel Costa 120221058 Engenharia Informática

Conteúdo

[Introdução 3](#_Toc469246694)

[Ambiente de desenvolvimento 3](#_Toc469246695)

[Software 3](#_Toc469246696)

[Requisitos 3](#_Toc469246697)

[Algoritmo geral 4](#_Toc469246698)

[Conteúdo da aplicação 4](#_Toc469246699)

[Algoritmos 4](#_Toc469246700)

[Breadth-First 4](#_Toc469246701)

[Depth-First 5](#_Toc469246702)

[A\* 5](#_Toc469246703)

[IDA\* 5](#_Toc469246704)

[Heurística 5](#_Toc469246705)

[Funções 6](#_Toc469246706)

[Análise dos resultados 7](#_Toc469246707)

[Resultados 7](#_Toc469246708)

[Quadro Comparativo 7](#_Toc469246709)

[Limitações da aplicação 7](#_Toc469246710)

[Conclusão 7](#_Toc469246711)

[Anexos 7](#_Toc469246712)

# Introdução

No âmbito da disciplina Inteligência Artificial, do curso de Engenharia informática do Instituto Politécnico de Setúbal – Escola Superior de Tecnologia de Setúbal, foi proposto pelos docentes da disciplina o desenvolvimento de um projeto, em LISP, denominado por “Puzzle dos pontos e caixas”, de modo que os alunos possam aplicar os seus conhecimentos adquiridos nas aulas práticas e laboratórios da disciplina.

Este jogo consiste em fechar um determinado número de caixas a partir de uma configuração inicial do tabuleiro. Para alcançar o objetivo será necessário desenhar um arco entre dois pontos adjacentes, tanto na horizontal como vertical, e quando o tiver 4 arcos desenhados, entre pontos adjacentes, temos uma caixa fechada.

Para mais informações sobre as características do jogo é favor consultar o enunciado do projeto de forma a integrar-se no jogo e o seu modo de procedimento.

# Ambiente de desenvolvimento

## Software

A Tabela 1 Softwares utilizados, contém os softwares utilizados para o desenvolvimento deste projeto:

Tabela Softwares utilizados

|  |  |
| --- | --- |
| LispWorks 7.0 (x64) | Ferramenta de desenvolvimento |
| Notepad++ | Editor de Texto |
| GIT | Repositório |

## Requisitos

A Tabela 2 Requisitos mínimos software, contém os requisitos mínimos e recomendados das máquinas de modo a suportar o LispWorks 7.0 (64bits):

Tabela Requisitos mínimos software

|  |  |
| --- | --- |
| Requisitos mínimos | |
| Windows (x64) | **Windows (x32)** |
| Processador AMD64|Intel 64  Memória de disco: 170MB  Sistema Operativo: Windows Vista SP2 (x64) | Processador Pentium 4  Memória de disco: 170MB  Sistema Operativo: Windows Vista SP2 (x64) |
| Macintosh (x64) | **Macintosh (x32)** |
| Processador Intel-based Macintosh  Memória de disco: 285MB  Sistema Operativo: OS X v.10.5 | Processador Intel-based Macintosh  Memória de disco: 265MB  Sistema Operativo: OS X v.10.5 |

# Algoritmo geral

## Conteúdo da aplicação

Este projeto foi proposto com a seguinte estrutura:

* Problemas.dat

Ficheiro que contêm os tabuleiros, isto é, os problemas que serão para resolver pelo programa. Cada problema é uma lista que contém uma lista de arcos verticais e arcos horizontais.

* Projeto.lisp

Ficheiro responsável pela interação com o utilizador, imprimindo menus no ecrã da máquina com as opções.

* Puzzle.lisp

Ficheiro responsável pela gestão das regras do domínio da aplicação. Neste ficheiro são encontrados os operadores, validações e outras funções úteis para o seu desenvolvimento.

* Procura.lisp

Neste ficheiro encontra-se a procura genérica e os algoritmos, assim como os sucessores, funções de cálculo (penetrância e fator de ramificação) e funções auxiliares. Responsável pela execução da procura de um problema até encontrar a solução objetiva.

**NOTA:**

Gostaríamos de alertar para a utilização dos ficheiros nos diversos Sistemas Operativos (Mac OS X; LINUX; Elementary entre outros), na qual não tínhamos qualquer meio de utilizar nesses mesmos ambientes.

## Algoritmos

Neste projeto foi proposto 4 algoritmos, na qual ao longo das próximas secções vamos descrever algumas das suas características.

### *Breadth-First*

A procura em largura (em Inglês Breadth-first) é um método de procura não-informada que expande todos os nós de uma árvore de uma solução. Este método realiza uma procura exaustiva numa árvore inteira, sem considerar o seu alvo de procura, até que ele o encontre. Este algoritmo não utiliza uma heurística.

Do ponto de vista do algoritmo, todos os nós filhos obtidos pela expansão de um nó são adicionados a uma fila (FIFO). Normalmente, nós que ainda não foram estudados pelos seus vizinhos são colocados num *container* (uma lista) designada por “lista de aberto". Uma vez examinados, são colocados num outro *container, a “lista de fechados”.*

### *Depth-First*

A procura em profundidade (*Depth-first* em Inglês) é um algoritmo de procura em profundidade realiza uma procura, não-informada, que progride através da expansão do primeiro nó filho da árvore de procura, e se aprofunda cada vez mais, até que o alvo da procura seja encontrado ou até que ele se depare com um nó que não possui filhos (nó folha). Então a procura retrocede (*backtrack*) e começa no próximo nó.

### A\*

A procura A\* é um tipo de procura informada (método heurística), ao contrário dos algoritmos já apresentados, que utiliza heurísticas de forma a direcionar a procura de uma solução. Recorrendo a uma heurística e estimando o custo de cada nó até a solução (composto por **g** (custo até ao nó em questão) e **h** (custo restante até à solução) é possível ordenar os nós a serem expandidos e assim explorar os que são mais prováveis de levar a uma solução. Desta forma reduz-se o número de nós a expandir tornando a pesquisa mais rápida.

### IDA\*

O IDA\* utiliza o mesmo conceito que o A\*, mas só que tenta reduzir a quantidade de memória utilizada através do uso de limiares de custo, **L.**

Isto é só mantém em memória, e expandem-se, nós que tenham um valor inferior ao limite estimado no nó inicial. Caso se chegue a um caso em que não se encontram nós que satisfaçam esta condição, volta a fazer executar a procura e escolhe um novo limiar, que assumo como limite o valor mais baixo por entre os nós descartados. Repete-se este procedimento até encontrar a solução.

**NOTA:**

Tanto o algoritmo A\* como IDA\* algoritmos encontram a solução ótima caso a heurística utilizada seja admissível.

# Heurística

As heurísticas acrescentam conhecimento sobre o domínio da aplicação aos algoritmos de procura informados (A\* e IDA\*). São regras empíricas que podem oferecer aceleração no processo de procura. As heurísticas são admissíveis quando, para qualquer grafo, descobre sempre o caminho ótimo até achar o objetivo. Neste projeto foi desenvolvido duas heurísticas, sendo uma proposta pelos docentes e outra ao nosso critério.

* Heuristica1

Fornecida pelo enunciado do projeto, consiste numa subtração do número de caixas fechadas de um tabuleiro pelo número de caixas ainda a fechar, escritas pelo utilizador e ainda subtrai 1. Esta heurística não é admissível, visto que o valor heurístico da solução não dá 0 (h=0). A forma apresentada será a seguinte:

* + - **h(x) = o(x) + c(x) – 1**, onde o(x) é o numero de caixas a fechar e c(x) o número de caixas fechadas de um determinado tabuleiro.
* Heuristica2

Heurística é proposta pelos alunos que realizaram o projeto onde consiste numa subtração do número de caixas fechadas de um tabuleiro pelo número de caixas ainda a fechar. Trata-se de uma heurística simples e eficaz e também é admissível, visto que para os problemas a solução apresenta um valor admissível (h=0). A forma apresentada será a seguinte:

* + - **h(x) = o(x) + c(x),** onde o(x) é o numero de caixas a fechar e c(x) o número de caixas fechadas de um determinado tabuleiro.

# Funções

Segue uma lista das funções principais, existentes em cada ficheiro, assim como a sua descrição:

* Projeto.lisp

|  |  |
| --- | --- |
| Função | Descrição |
| (iniciar) | Função que compila os ficheiros e inicializa-os juntamente com o menu inicial. |
| (menu-inicial) | Apresenta o menu principal do programa. |
| (iniciar-procura) | Função que pede ao utilizador toda a informação necessária para começar uma procura no espaço de estados. |
| (ler-tabuleiro) | Lista todos os problemas e faz a leitura numa diretoria que chama o ficheiro que contém todos os problemas. |
| (ler-numero-objectivo-caixas) | Lê o numero de caixas que o utilizador pretende fechar. |
| (ler-algoritmo) | Função que recebe do utilizador o algoritmo que pretende utilizar do problema anteriormente escolhido. |
| (ler-heuristica) | No caso do utilizador escolher o algoritmo a-asterisco ou ida-asterisco, será mostrado o menu que mostra as duas heurísticas que pretende usar na procura |
| (ler-profundidade) | Função que lê a profundidade máxima do utilizador caso este escolha o algoritmo depth-first (procura em profundidade). |
| (directoria-atual) | Define um caminho para ler os ficheiros. |
| (resultados) | Imprime os resultados do jogo. |

* Procura.lisp

|  |  |
| --- | --- |
| Função | Descrição |
| (procura-generica) | Permite procurar a solução de um problema usando a procura no espaço de estados. A partir de um estado inicial,  de uma função que gera os sucessores e de um dado algoritmo |
| (bfs) | Função que executa o algoritmo de procura em largura. Faz uma junção da lista de abertos com os sucessores. |
| (dfs) | Função que executa o algoritmo de procura em profundidade. Faz uma junção da lista de sucessores com os abertos. |
| (a-asterisco) | Função que executa o algoritmo ida\*. Faz uma junção da lista de sucessores com os abertos, tendo em conta a ordenação do valor do custo. |
| (ida-asterisco) | - |
| (sucessores) | Função que retornada uma lista com todos os sucessores de um nó. |
| (existep) | Retorna valor verdadeiro, se o nó existir na lista. Para o algoritmo dfs, o conceito de nó repetido é particular. |
| (existe-solucao) | Verifica se existe uma solução ao problema numa lista de sucessores para o algoritmo dfs |
| (penetrancia) | Retorna o valor da penetrância dos nos gerados até o nó objetivo sobre dos nos totais gerados. |
| (fator-ramificacao) | - |

* Procura.lisp

|  |  |
| --- | --- |
| Função | Descrição |
| (criar-no) | Cria uma lista que representa um nó. Um nó é composto pelo estado que é o tabuleiro, este é um parâmetro obrigatório, é composto também por outros parâmetros, como a profundidade a que se encontra, pela heurística deste mesmo nó e pelo nó pai, ou seja, o nó que o gerou.  A profundidade e a heurística por omissão têm valor nil, enquanto que o nó pai por defeito é NIL. |
| (get-no-estado) | Retorna o estado do nó, que é representado pelo tabuleiro. |
| (get-no-profundidade) | Retorna a profundidade em que o nó se encontra. |
| (get-no-heuristica) | Retorna a heurística do nó. |
| (get-no-pai) | Retorna o nó pai deste nó, ou seja, o nó que gerou este nó. |
| (custo) | Retorna o valor do custo do nó (f). Soma do valor da profundidade com o valor heurístico. |
| (get-arcos-horizontais) | Retorna a lista dos arcos horizontais de um tabuleiro. |
| (get-arcos-verticais) | Retorna a lista dos arcos verticais de um tabuleiro. |
| (Inserir-arco-na-posicao) | Insere um arco (representado pelo valor [T]) numa lista que representa o conjunto de arcos dum tabuleiro. A posição representada pela linha e a coluna de destino são valores inteiros passados como argumentos. |
| (inserir-arco-vertical) | Insere um arco vertical (representado pelo valor [T]) num tabuleiro passado como argumento. |
| (inserir-arco-horizontal) | Insere um arco horizontal (representado pelo valor [T]) num tabuleiro passado como argumento. |
| (possível-adicionar-arco) | Recebe índices de linha e coluna e uma lista de arcos horizontais ou de arcos verticais e verifica se naquela posição o valor é [T], se for devolve [NIL], se for [NIL] devolve [T]. |
| (operadores) | Cria uma lista com todos os operadores do problema dos Pontos e das Caixas. |
| (numero-linhas-tabuleiro) | Retorna o número de linhas verticais de um tabuleiro. |
| (numero-colunas-tabuleiro) | Retorna o número de linhas verticais de um tabuleiro. |
| (caixas-fechadas) | Retorna o numero de caixas fechadas de um tabuleiro. |
| (contar-objetivo) | Função que irá contar se a caixa está fechada ou não, isto é, se a função auxiliar conta-caixa-fechada |
| (contar-nils-lista) | Função que irá contar o numero de NILS's existentes numa lista.Se não existir dará valor 0, ou seja, teriamos uma lista so com T,o que resulta uma caixa fechada. |
| (heuristica1) | Usada uma heurística que privilegia os tabuleiros com o maior número de caixas fechadas. |
| (heuristica2) | Heurística efetuada pelos alunos que se baseia na heuristica1. |
| (solucaop) | Devolve [T] se o número de caixas a fechar for igual ao número de caixas fechadas do nó, e devolve [NIL] se não for |
| (caminho-solucao) | Retorna o caminho até à solução. Ou seja, todos os estados desde a inicial à solução. |

# Análise dos resultados

## Resultados

Os resultados seguintes, serão mostrados por ordem do problemas:

|  |  |
| --- | --- |
|  | Problema A |
| Breadth-First |  |
| Depth-First |  |
| A\* - heuristica fornecida pelo docente |  |
| A\* - heuristica criada pelo aluno |  |
| IDA\* - heuristica fornecida pelo docente | - |
| IDA\* - heuristica criada pelo aluno | - |

|  |  |
| --- | --- |
|  | Problema B |
| Breadth-First |  |
| Depth-First |  |
| A\* - heuristica fornecida pelo docente |  |
| A\* - heuristica criada pelo aluno |  |
| IDA\* - heuristica fornecida pelo docente | - |
| IDA\* - heuristica criada pelo aluno | - |

|  |  |
| --- | --- |
|  | Problema C |
| Breadth-First |  |
| Depth-First |  |
| A\* - heuristica fornecida pelo docente |  |
| A\* - heuristica criada pelo aluno |  |
| IDA\* - heuristica fornecida pelo docente | - |
| IDA\* - heuristica criada pelo aluno | - |

|  |  |
| --- | --- |
|  | Problema D |
| Breadth-First |  |
| Depth-First |  |
| A\* - heuristica fornecida pelo docente |  |
| A\* - heuristica criada pelo aluno |  |
| IDA\* - heuristica fornecida pelo docente | - |
| IDA\* - heuristica criada pelo aluno | - |

|  |  |
| --- | --- |
|  | Problema E |
| Breadth-First |  |
| Depth-First |  |
| A\* - heuristica fornecida pelo docente |  |
| A\* - heuristica criada pelo aluno |  |
| IDA\* - heuristica fornecida pelo docente | - |
| IDA\* - heuristica criada pelo aluno | - |

|  |  |
| --- | --- |
|  | Problema F |
| Breadth-First |  |
| Depth-First |  |
| A\* - heuristica fornecida pelo docente |  |
| A\* - heuristica criada pelo aluno |  |
| IDA\* - heuristica fornecida pelo docente | - |
| IDA\* - heuristica criada pelo aluno | - |

## Quadro Comparativo

No quadro seguinte (Tabela 3 Quadro comparativo dos resultados), podemos ter uma ideia do tempo de execução para cada problema, utilizando algoritmos diferentes:

Tabela Quadro comparativo dos resultados

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Problemas | Breadh-First | Depth-First | A\* (heuristica1) | A\* (heuristica2) | IDA\* (heuristica1) | IDA\* (heuristica2) |
| A | 0 | 0 | 0 | 0 | - | - |
| B | 0 | 0 | 0 | 0 | - | - |
| C | - | 0 | - | - | - | - |
| D | - | 0 | - | - | - | - |
| E | - | 0 | - | - | - | - |
| F | - | 0 | - | - | - | - |

# Limitações da aplicação

Neste projeto, existe uma limitação relativamente ao *software* LisWorks, ou seja, é limitado. Caso não fosse limitado, isto é, tivesse mais memória, o programa encontrava solução dos problemas em causa.

# Conclusão

Com o desenvolvimento deste primeiro projeto, ao longo das últimas semanas foi bastante enriquecedor no que toca aos assuntos abordados de LISP, e são sem dúvida uma mais valia na futura vida académica e profissional, no entanto, e de notar que é uma linguagem que apresenta um grau de dificuldade exigente.

# Anexos