Engenharia Informática

Inteligência Artificial

MONTA-CARGAS

Luís Matos

Nº 2151361

2151361@my.ipleiria.pt

Mayelson de Sousa

Nº 2151582

2151582@my.ipleiria.pt

**ABSTRACT**

No âmbito da cadeira de Inteligência Artificial foi proposto a elaboração de um programa que resolva puzzles do jogo Monta-Cargas, recorrendo a algoritmos de procura informados e não informados. Este artigo descreve a maneira como implementámos a resolução do jogo.

**ENVIRONMENT**

Este projeto foi um implementado no Netbeans e compilado para o Java 1.8. O Computador usado para os testes fui um Acer Aspire 5741ZG com um processador Pentium(R) CPU P6000 a 1.87GHz, 2 Cores, com 4.0 GB de RAM e com o Windows 10 64 bits.

**ALGORITHMS**

Neste projeto usámos os algoritmos de pesquisa para resolução de problemas, entre os quais:

Algoritmos Não Informados:

* Breadth First Search
* Depth First Search
* Depth Limited First Search

Algoritmos Informados:

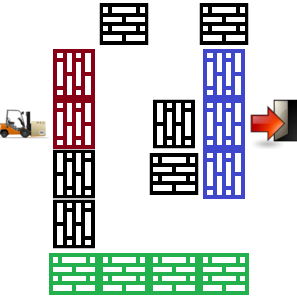
* A\* Search
* IDA\* Search
* Beam Search
* Greedy Best First Search
* Iterative Deepening Search
* Uniform cost search

**MONTA-CARGAS**

O MONTA-CARGAS é baseado no Mouse um jogo para a mobile em que o objetivo consiste em descobrir a sequência de ações que permita deslocar um conjunto de caixas de modo a desobstruir o caminho entre o rato e a sua toca presenteando o jogador com a pontuação máxima caso este consiga descobrir a solução com o menor numero de ações possível.

No MONTA-CARGAS o conceito é o mesmo apenas substituímos o rato por um monta-cargas e a toca pela porta do armazém. A figura seguinte mostra uma possível configuração inicial do jogo. A saída encontra-se sempre do lado direito e o monta-cargas só se pode mover para a esquerda ou para a direita. O Armazém contém um conjunto de caixas que só se podem mover em linha reta dependendo da sua orientação (verticalmente ou horizontalmente).

Em cada estado só se pode mover uma peca do jogo para uma posição adjacente caso essa posição não esteja ocupada por outra peca, à exceção do monta-cargas, pois este pode mover-se para cima da porta terminando assim o jogo.



**Figura 1.** Monta-Cargas

# INTRODUCTION

Pretende-se implementar um programa que resolva os puzzles do Monta-Cargas, recorrendo aos algoritmos de pesquisa implementados nas aulas (algoritmos de pesquisa não informados e informados). O objetivo é usar estes algoritmos de forma a chegar à melhor solução para a resolução do problema. Pretende-se também formular heurísticas que permitam resolver os puzzles de uma maneira mais otimizada.

Para além da implementação dos algoritmos pedidos para a resolução do problema, pretende-se que seja também realizado um estudo comparativo do desempenho dos vários algoritmos de pesquisa bem como das heurísticas formuladas para a resolução dos puzzles.

# IMPLEMENTATION

Como base para este projeto usamos o projeto do Puzzle-Eigth que foi desenvolvido nas aulas práticas. Este projeto já incluía a interface gráfica, os algoritmos de procura implementados e um agente que faz uso destes algoritmos para resolver o Puzzle-Eigth. Vamos agora explicar alterações e adaptações que fizemos para transformar este projeto no projeto proposto.

* 1. First Steps

Começamos por modificar a interface gráfica expandimos a grelha de 3x3 para 6x6 (este tamanho é agora definido no ficheiro do problema) e alterámos as imagens dos números para imagens de pecas do MONTA-CARGAS.

Em relação as classes existentes para resolver o problema decidimos fazer *Refracting* em todas as classes que tinham a ver com a resolução do Puzzle-Eight isto também serviu para ter uma ideia inicial de como estas classes interagiam umas com as outras.

* 1. New Classes
     1. Peca

Ao analisar o problema categorizamos todas as entidades do problema que se podem mover como uma peca, por isso criamos a classe abstrata Peca, esta classe tem apenas 4 atributos, linha, coluna, tamanho e uma lista de ações. Agora podemos mencionar o carro ou uma caixa como peca com posição (linha e coluna), tamanho e ações.

A posição de uma peca é obtida através da sua localização na matriz do estado já o seu tamanho e ações são atributos que dependem da peca em questão por isso definimos uma classe para cada peca que estende esta classe.

Cada peca apenas tem 2 ações, ou se pode mover para cima e para baixo (Verticalmente) ou para a direita e para a esquerda (Horizontalmente). As classes destas ações já existiam no projeto base.

* + 1. Posicao

Esta classe apenas possui dois inteiros que são o i (linha) e j (coluna). Esta classe fui criada para evitar ler a mesma peca da matriz caso esta já tenha sido criada.

Por exemplo imaginemos que uma linha da matriz é a seguinte [0 0 6 6 6 0], ao lermos o primeiro 6 já sabemos que nessa posição está uma Caixa com tamanho 3 na vertical e não precisamos de ler os próximos 2 números que se seguem por isso adicionamos essas próximas posições a uma lista de posições a passar quando estamos a ler a matriz.

Esta classe também podia ter sido usada como um atributo da classe Peca mas decidimos não o fazer para obtermos código mais limpo.

* 1. New Attributes and Methods on the State
     1. LinkedList<Peca> pecas;

Este atributo armazena todas as pecas do estado atual do problema e é preenchido quando analisamos a matriz que foi passada para construir o estado atual.

Como as pecas nunca são destruídas ou novas pecas são criadas criamos um construtor para o State que recebe uma lista deste tipo e copia essa lista para este atributo, assim não precisamos de analisar novamente a matriz caso já exista um estado anterior.

* + 1. Boolean canMove(Peca peca) & void move(Peca peca)

Para cada direção que uma peca se pode mexer existe o par de métodos canMove(Peca peca) e move(Peca peca) uma serve para verificar se a peca em questão se pode mover e a outra para mover a peca repectivamente, por exemplo no caso da actionRight existem os métodos canMoveRight(Peca peca) e moveRight(Peca peca).

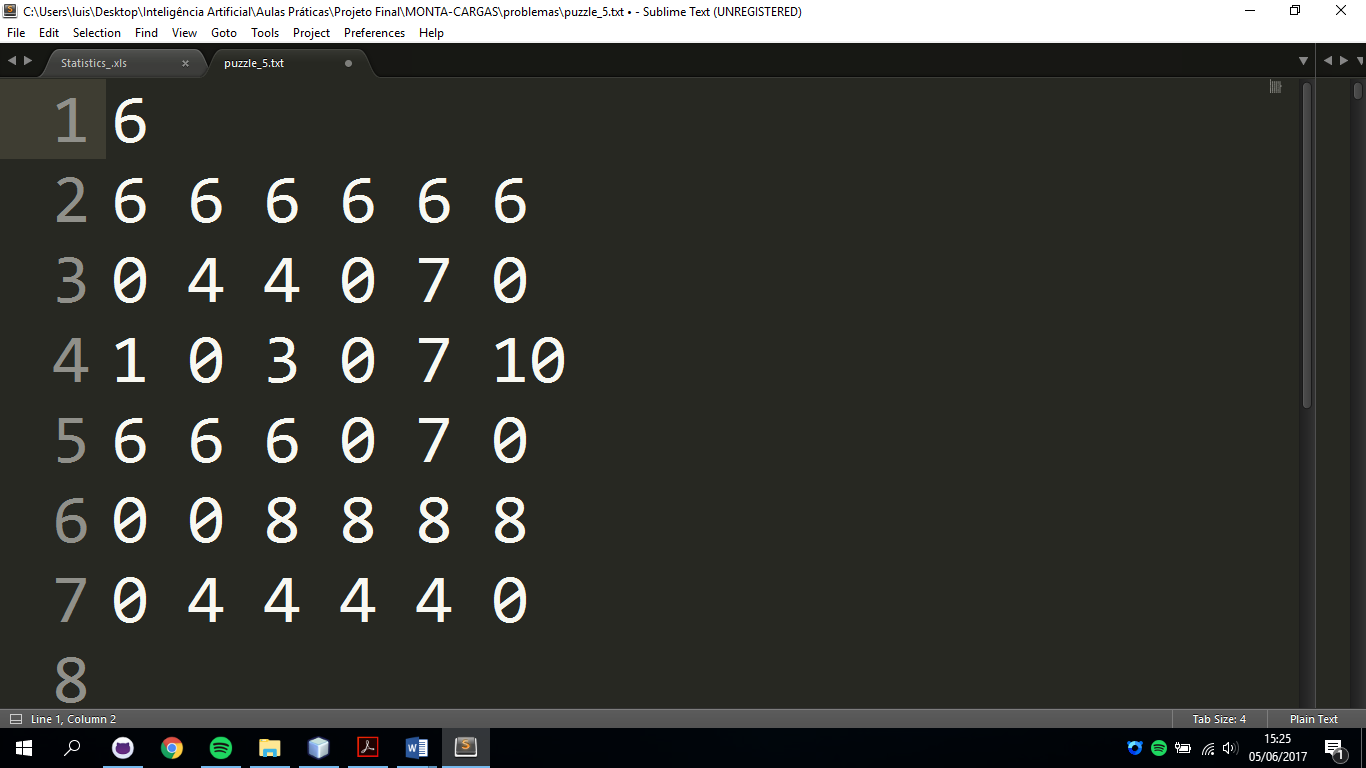
O caso do canMoveRight é um caso particular, pois se a peca for o carro este pode mover-se para a localização da porta.

* + 1. LinkedList<Action> getActions()

Este método é o responsável por devolver ao problema todas as ações que cada peca do estado atual detém.

* 1. State

Cada estado é definido por uma matriz de inteiros em que o numero 1 representa o carro e o numero 10 representa a porta, qualquer outro numero representa uma caixa especifica a esse número. A figura seguinte mostra o estado inicial do puzzle 5.



**Figura 2.** Estado inicial o puzzle 5.

* 1. Observations

Como já devem ter reparado a porta não é considerada uma peça. A porta esta apenas é mostrada na grelha e a sua posição é calculada consoante a coluna do carro. No caso do carro se encontrar na posição da porta significa que uma solução foi encontrada.

Todos os puzzles para este problema foram fornecidos pelos docentes desta disciplina.

# Performance

Neste capítulo vamos abordar os testes que fizemos com os diferentes algoritmos e comparar os resultados mais relevantes.

Estes testes foram realizados nos puzzle 1, que é o mais simples de todos.

* 1. Performance of Not Informed Search Algorithms

Como podemos observar pelos gráficos acima, os métodos de procura não informados mais promissores são o Uniform Cost e o Breadth First pois encontram a solução com menor custo e um numero reduzido de nós explorados.

Vamos realçar que o Depth First e o Depth Limited First não encontraram a melhor solução e o Iterative Deepening explorar uma quantidade absurda de nós em comparação com os outro algoritmos.

* 1. Performance of Informed Search Algorithms

Nos algoritmos de procura informada todos eles encontram a solução com o menor custo, por isso vamos apenas analisar a quantidade de nos explorados.

A Heurística usada para estas comparações foi a “Distance to Door and Tiles Size In Front of Car”

# HEURISTICS

* 1. Analysis

A cada estado conseguimos obter várias informações que, futuramente ajudaram a desenvolver várias heurísticas. Ao olharmos com atenção, há 3 dados importantes e que devem ser trabalhados de modo a encontrar a heurística certa:

* Número de caixas entre o carro e a porta
* Distância do carro à porta
* O tamanho das peças
  1. Descriptions of used Heuristics

The title (Helvetica 18-point bold), authors' names (Helvetica 12-point) and affiliations (Helvetica 10-point) run across the full width of the page – one column wide. We also recommend phone number (Helvetica 10-point) and e-mail address (Helvetica 12-point). See the top of this page for three addresses. If only one address is needed, center all address text. For two addresses, use two centered tabs, and so on. For more than three authors, you may have to improvise.[[1]](#footnote-1)

* 1. First Page Copyright Notice

Please leave 3.81 cm (1.5") of blank text box at the bottom of the left column of the first page for the copyright notice.

* 1. Subsequent Pages

For pages other than the first page, start at the top of the page, and continue in double-column format. The two columns on the last page should be as close to equal length as possible.

Table 1. Table captions should be placed above the table

|  |  |  |  |
| --- | --- | --- | --- |
| **Graphics** | **Top** | **In-between** | **Bottom** |
| Tables | End | Last | First |
| Figures | Good | Similar | Very well |

* 1. References and Citations

Footnotes should be Times New Roman 9-point, and justified to the full width of the column.

Use the standard Communications of the ACM format for references – that is, a numbered list at the end of the article, ordered alphabetically by first author, and referenced by numbers in brackets [1]. See the examples of citations at the end of this document. Within this template file, use the style named references for the text of your citation.

The references are also in 9 pt., but that section (see Section 7) is ragged right. References should be published materials accessible to the public. Internal technical reports may be cited only if they are easily accessible (i.e. you can give the address to obtain the report within your citation) and may be obtained by any reader. Proprietary information may not be cited. Private communications should be acknowledged, not referenced (e.g., “[Robertson, personal communication]”).

* 1. Page Numbering, Headers and Footers

Do not include headers, footers or page numbers in your submission. These will be added when the publications are assembled.

# FIGURES/CAPTIONS

Place Tables/Figures/Images in text as close to the reference as possible (see Figure 1). It may extend across both columns to a maximum width of 17.78 cm (7”).

Captions should be Times New Roman 9-point bold. They should be numbered (e.g., “Table 1” or “Figure 2”), please note that the word for Table and Figure are spelled out. Figure’s captions should be centered beneath the image or picture, and Table captions should be centered above the table body.

# SECTIONS

The heading of a section should be in Times New Roman 12-point bold in all-capitals flush left with an additional 6-points of white space above the section head. Sections and subsequent sub- sections should be numbered and flush left. For a section head and a subsection head together (such as Section 3 and subsection 3.1), use no additional space above the subsection head.



Figure 1. Insert caption to place caption below figure.

.

* 1. Subsections

The heading of subsections should be in Times New Roman 12-point bold with only the initial letters capitalized. (Note: For subsections and subsubsections, a word like *the* or *a* is not capitalized unless it is the first word of the header.)

* + 1. Subsubsections

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized and 6-points of white space above the subsubsection head.

* + - 1. Subsubsections

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized.

* + - 1. Subsubsections

The heading for subsubsections should be in Times New Roman 11-point italic with initial letters capitalized.

# ACKNOWLEDGMENTS

Our thanks to ACM SIGCHI for allowing us to modify templates they had developed.

# REFERENCES

1. Bowman, B., Debray, S. K., and Peterson, L. L. Reasoning about naming systems. *ACM Trans. Program. Lang. Syst., 15,* 5 (Nov. 1993), 795-825.
2. Ding, W., and Marchionini, G. *A Study on Video Browsing Strategies.* Technical Report UMIACS-TR-97-40, University of Maryland, College Park, MD, 1997.
3. Fröhlich, B. and Plate, J. The cubic mouse: a new device for three-dimensional iput. In *Proceedings of the SIGCHI conference on Human factors in computing systems   
   (CHI ’00)* (The Hague, The Netherlands, April 1-6, 2000). ACM Press, New York, NY, 2000, 526-531.
4. Lamport, L. *LaTeX User’s Guide and Document Reference Manual.* Addison-Wesley, Reading, MA, 1986.
5. Sannella, M. J. *Constraint Satisfaction and Debugging for Interactive User Interfaces.* Ph.D. Thesis, University of Washington, Seattle, WA, 1994.

Columns on Last Page Should Be Made As Close As Possible to Equal Length

1. If necessary, you may place some address information in a footnote, or in a named section at the end of your paper. [↑](#footnote-ref-1)