

# Manual Técnico - 1ª Fase do Projeto

---



**POLITECNICO  
SETUBAL**

## UC - Inteligência Artificial

Docentes:

- Prof. Joaquim Filipe
- Eng. Filipe Mariano

## Jogo do Cavalo

---

Projeto realizado por Grupo 51:

- Miguel Neves - 201900377
- Guilherme Ravasco - 201900646

# Índice

---

1. Introdução
2. Arquitetura do Sistema
3. Entidades
4. Algoritmos
5. Resultados
6. Limitações técnicas

## 1. Introdução

---

Este projeto tem como objetivo realizar um programa para resolver o **Jogo do Cavalo** com alguns algoritmos de busca desenvolvidos em **Common Lisp**. Os algoritmos de busca utilizados para resolver este jogo foram, o BFS (Procura em Largura), o DFS (Procura em Profundidade) e o algoritmo A\*. O **Jogo do Cavalo** é uma variante do problema matmático conhecido como o **Passeio do Cavalo**, onde o objetivo é, através dos movimentos do cavlo, conseguir visitar todas as casas de um tabuleiro parecido ao de xadrez. Esta versão decorrerá num tabuleiro de 10 linhas e 10 colunas (10x10), onde cada casa possui uma pontuação. Estes movimentos são implementados como operadores em **LISP**. O projeto propõe a resolução de problemas de A a F, adicionalmente tem um problema G, que é fornecido pelo docente na hora da discussão do projeto, que estão anexados no final do enunciado do projeto, cada um representa um desafio para o **Jogo do Cavalo**. Para conseguir a solução mais eficiente de cada problema, são utilizados os algoritmos de busca, a procura em largura explora o espaço de estados, a procura em profundidade explora as possibilidades em profundidade e o algoritmo A\* utiliza uma heurística para conseguir guiar a procura pelo resultado mais eficiente.

## 2. Arquitetura do Sistema

---

O projeto desenvolvido para a resolução do **Jogo do Cavalo** foi implementado em **Common Lisp** que foi a linguagem lecionada ao longo da Unidade Curricular de Inteligência Artificial, utilizando o **Lisp Works** como ferramenta de desenvolvimento de código.

A resolução do **Jogo do Cavalo** está subdividido em 4 ficheiros:

- puzzle.lisp - Implementação das funções para a resolução do problemas, operadores e heurística.
- procura.lisp - Implementação dos algoritmos de procura juntamente com os seletores dos nós.
- projeto.lisp - Interação com o utilizador e leitura de ficheiros.
- problemas.dat - Problemas em anexo do enunciado de A a F, e adicionalmente o problema G que é fornecido pelo docente na hora da avaliação.

## 3. Entidades

---

### Tabuleiro

O tabuleiro é representado em forma de lista de listas em **Common LISP**, o tabuleiro é composto por 100 átomos, pois o tabuleiro é composto por 10 linhas e 10 colunas, cada átomo representa uma casa, e estas têm um valor numérico de 0 a 99.

### Funções e Regras do Jogo

Nesta parte do manual técnico estão descritas as funções e regras que foram implementadas no projeto, sabendo que as movimentações do cavalo apenas podem ser realizadas para casas disponíveis no tabuleiro.

#### Regra do Cavalo

Esta regra tem como objetivo principal colocar o cavalo no jogo, a regra é aplicada sempre que se começa a procura pelo tabuleiro. Esta regra começa por validar se o cavalo está posicionado numa casa e verifica se este está na primeira linha do tabuleiro.

O resultado desta regra é o conjunto de casas que estão disponíveis para realizar a jogada, aplicando também as regras do Simétrico e do Duplo que estão descritas abaixo.

#### Regra do Simétrico

Esta regra coloca o valor simétrico da casa visitada no movimento atual como **NIL**, após isto, essa mesma casa já não poderá ser mais visitada em jogadas futuras. Um número simétrico é por exemplo temos o valor "**34**" o seu simétrico será "**43**".

#### Regra do Duplo

Esta regra verifica se o valor da casa que está a ser visitada no movimento da jogada atual é um número duplo. Por exemplo, "**33**", "**55**". O que esta regra irá fazer com este número duplo é colocar na casa com o maior número duplo disponível a **NIL**, esta casa após isto não poderá ser visitada em jogadas futuras.

### Representação de Estados

O **Jogo do Cavalo** é um desafio que envolve encontrar o caminho de um cavalo num tabuleiro de xadrez, passando por todas as casas exatamente uma vez. Para resolver este problema, é fundamental abordá-lo em termos de estados, representando as diferentes configurações do tabuleiro em cada etapa do jogo. Além disso, é preciso definir operadores que permitem a transição entre estes estados, levando assim o cavalo da sua posição inicial até à posição final.

O problema pode ser abordado utilizando técnicas de busca em árvore, explorando assim as várias possibilidades de movimento do cavalo a partir de cada estado atual.

## Operadores

Baseando-nos nas oito operações disponíveis em L, criamos oito operadores que executam esses mesmos movimentos. Essas operações podem ser realizadas em relação às linhas e às colunas do tabuleiro.

Na organização da função, a representação é feita por coordenadas (coluna, linha), onde a coluna corresponde aos movimentos para cima e para baixo, e a linha representa os movimentos para a direita e para a esquerda.

Os operadores representam os seguintes movimentos:

- operador-1 (2 1)
- operador-2 (1 2)
- operador-3 (-1 2)
- operador-4 (-2 1)
- operador-5 (-2 -1)
- operador-6 (-1 -2)
- operador-7 (1 -2)
- operador-8 (2 -1)

## Nó

O nó nos algoritmos **BFS** e **DFS** é constituído por, (**Tabuleiro|Profundidade|Pontos|Operadores**). O nó no algoritmo **A\*** é constituído por, (Tabuleiro|Profundidade|Pontos|Operadores|Heurística|Valor do tabuleiro Pai)

## Sucessões

A Sucessão é um novo nó gerado a partir do estado do nó anterior.

# 4. Algoritmos

---

Neste projeto, enfrentamos o desafio do **Jogo do Cavalo**, onde o principal objetivo é alcançar uma determinada pontuação em cada problema com o menor número possível de movimentos. Para resolver este desafio, aplicamos os algoritmos discutidos em aula, utilizando-os para guiar o cavalo ao longo do tabuleiro. O nosso foco centra-se na busca pela solução através da exploração dos movimentos possíveis até que não seja mais possível alcançar os objetivos propostos.

A condição de paragem para os algoritmos implementados é determinada por duas condições: quando o cavalo não possui mais movimentos disponíveis ou quando o objetivo de pontuação foi alcançado.

## DFS

O algoritmo **Depth-First-Search**, como o seu nome indica faz a procura em profundidade nos nós do tabuleiro, neste algoritmo o cavalo vai-se deslocar de uma casa para outra, explorando todos os nós à medida que vai avançando. Os nós explorados tornam-se os sucessores e são adicionados à frente da fila de nós abertos, destacando a expansão vertical na busca. Isto significa que o algoritmo dá prioridade à função de percorrer o máximo possível numa direção antes de recuar e explorar outras opções.

## BFS

O algoritmo **Breath-First-Search**, como o seu nome indica realiza uma procura em largura dos estados, este começa no nó da raiz e segue em direção ao nó de solução. Durante este processo de procura, o algoritmo examina os nós vizinhos que estão na profundidade atual do tabuleiro, evitando assim revisitar o nó pai que já foi visto anteriormente.

## A\*

O algoritmo **A\*** difere dos algoritmos anteriores no que toca à busca dos nós numa árvore. Enquanto os algoritmos **BFS** e **DFS** concentram-se essencialmente na pesquisa de estados num espaço com um problema menos complexo, o algoritmo **A\*** permite a busca num espaço com um problema mais complexo.

No algoritmo **A\***, uma árvore de sucessores é gerada utilizando uma função heurística para calcular o custo. Esse custo é essencial para ordenar as listas dos nós que representam as possíveis jogadas no tabuleiro. Esta abordagem visa encontrar o nó de possível solução da maneira mais eficiente.

## Heurística

A utilização de heurísticas permite conduzir a pesquisa ao quantificar a proximidade de um determinado objetivo. Neste projeto utilizamos duas heurísticas, uma dada e uma criada pelo grupo, para ser possível encontrar as possíveis soluções consoante os objetivos de cada tabuleiro.

### Heurística Base

Tal como está escrito no enunciado fornecido, utilizámos a heurística dada, esta heurística visita as casas com a maior pontuação.

$$H(x) = O(x) / M(x)$$

$M(x)$  é a média por casa dos pontos que constam no tabuleiro  $x$ .

$O(x)$  é o número de pontos que faltam para atingir o valor definido como objetivo.

### Heurística Criada

$$H(x) = O(x) / M(x)$$

$O(x)$  pontos capturados na jogada.

$M(x)$  pontos disponíveis no tabuleiro pai.

## Ordenação

Para odernar a lista de nós foram desenvolvidas algumas funções para ajudar a completar os algoritmos:

### Ordenação BFS (abertos-bfs)

Esta função serve para atualizar a lista de nós abertos durante a exploração da árvore de busca. Ao adicionar os nós sucessores à frente da lista de nós abertos, o **BFS** continua a explorar os nós numa ordem que garante a busca em largura.

Ordenação DFS (abertos-dfs)

Esta função é relativamente parecida à anterior, a única diferença é que esta função explora os nós numa ordem que favorece a profundidade da busca.

Ordenação A\*

- sort-a-star: esta função classifica os nós em ordem decrescente com base na sua heurística.
- abertos-a-star: esta função serve para atualizar a lista de nós abertos durante a exploração da árvore de busca. Ordenar os nós abertos com base na heurística dos mesmos ajuda o algoritmo **A\*** a priorizar a exploração dos nós mais promissores primeiro.

5. Resultados

Para poder comparar a eficácia dos 3 algoritmos funcionais foi feito uma tabela com as estatísticas de cada algoritmo na resolução de cada problema.

**ABF( Average Branching Factor )** -> Fator Médio de Ramificação

BFS (Breadth First Search)

Problema	Profundidade	Nós Gerados	Nós Expandidos	Elapsed Time	Pontuação	ABF	Penetrância
A	0	7	6	0.0	72	1.1428572	0.42857143
B	8	41	39	0.058	60	1.175	0.19512195
C	6	31	25	0.041	272	1.6923077	0.19354838
D	10	180	166	22.254	643	1.4251497	0.0555(5)
E stack overflow							
F stack overflow							

DFS (Depth-first search)

Problema	Profundidade	Nós Gerados	Nós Expandidos	Elapsed Time	Pontuação	ABF	Penetrância
A	3	5	4	0.0 08	72	1.2	0.6
B	8	33	32	0.050	60	1.2121213	0.24242425
C	7	11	6	0.009	302	2.0	0.6363636
D	12	23	12	0.025	657	2.0	0.5217391
E stack overflow							

Problema	Profundidade	Nós Gerados	Nós Expandidos	Elapsed Time	Pontuação	ABF	Penetrância
F	40	151	39	0.108	2003	3.775	0.26490065

A\* (A\* Search Algorithm) - Heurística Base

Problema	Nós Gerados	Nós Expandidos	Profundidade	Heurística $h(x)$	Penetrância	Pontuação	ABF
A	8	7	3	$h(x)=o(x)/m(x)$ -0.08163265	0.375	72	1.1428572
B Solução não encontrada							
C Solução não encontrada							
D Solução não encontrada							
E Solução não encontrada							
F stack overflow							

A\* (A\* Search Algorithm) - Heurística Criada

Problema	Nós Gerados	Nós Expandidos	Profundidade	Heurística $h(x)$	Penetrância	Pontuação	ABF
A	5	3	3	$h(x)=o(x)/m(x)$ 0.30985916	0.6	72	1.25
B	23	17	10	$h(x)=o(x)/m(x)$ 1.0	0.4347826	65	1.5625
C stack overflow							
D solução não encontrada							
E solução não encontrada							
F	76	54	29	$h(x)=o(x)/m(x)$ 0.01998645	0.38157895	2057	1.555(5)

## 6. Limitações Técnicas

---

Ao longo do desenvolvimento do projeto foram encontradas várias dificuldades, como, gerar os nós do **A\*** com dados para as estatísticas, o algoritmo **BFS** tem dificuldades em resolver os problemas E e F do anexo do enunciado, o algoritmo **DFS** tem dificuldade em resolver o problema E, devido à memória stack limitada pela versão grátis do IDE. As limitações técnicas encontradas no algoritmo **A\*** com a heurística Base são, problema em encontrar solução para o problema B, C, D e E, para o problema F, o IDE dá stack overflow, para a heurística criada, o algoritmo **A\*** dá stack overflow no problema C, não encontra solução para o problema D e problema E.