Manual Técnico - Projeto Jogo do Solitário

Inteligência Artificial 2025/26

Prof. Joaquim Filipe

Eng. Filipe Mariano

Autores

- António Guerreiro (202200160)
- Bruno Leite (202100504)
- Guilherme Cruz (2024129841)

Data: 12 de dezembro de 2025

---

1. Introdução

No âmbito da unidade curricular de Inteligência Artificial, este projeto implemente e compara algoritmos de procura (BFS, DFS, A*) aplicados à resolução do problema do **Jogo do Solitário** (Peg Solitaire). O objetivo do jogo é reduzir o número de pinos no tabuleiro para um, através de movimentos de salto.

Este manual técnico descreve detalhadamente a arquitetura do sistema, as entidades, as funções implementadas e os algoritmos de procura utilizados, todos desenvolvimento em Common LISP.

---

2. Arquitetura do Sistema

O projeto está estruturado em três ficheiros principais de código LISP e um ficheiro de dados:

| Ficheiro | Conteúdo Principal | Fontes |

| `projeto.lisp` | Carrega módulos, trata da interação com o utilizador (**menus**), leitura de problemas (`problemas.dat`) e escrita de logs de resultados. | `puzzle.lisp` | Contém a lógica específica do problema (domínio): representação do tabuleiro, operadores de movimento e funções heurísticas. | `procura.lisp` | Implementa o Tipo Abstrato de Dados **Nó** e os algoritmos de procura: **BFS**, **DFS** e **A***. | `problemas.dat` | Contém os estados iniciais dos tabuleiros para os problemas (A a F).

---

3. Entidades e Implementações (Módulo `puzzle.lisp`)

3.1. Representação do Tabuleiro

O tabuleiro de Solitário Britânico (7x7) é representado como uma **lista de listas**.

- Os elementos `1` representam um pino (peça).
- Os elementos `0` representam um buraco vazio.
- Os elementos `nil` representam posições inválidas no formato do tabuleiro.

3.2. Funções de Domínio Chave

Função | Descrição | Fontes |

| `celula` | Retorna o valor na posição `(l, c)` (1-indexado). Retorna `nil` se fora dos limites. | `substituir` | Retorna um **novo tabuleiro** com a célula `(l,c)` alterada. | `aplicar-movimento` | Aplica a lógica do salto: **Origem (1) \(\rightarrow\) 0, Meio (1) \(\rightarrow\) 0, Destino (0) \(\rightarrow\) 1**. | `solucaop` | Verifica se o nó é solução, contando se resta **apenas 1 pino** no estado.

3.3. Operadores de Movimento

- `operador-cd`: Captura Direita (`l, c` salta para `l, c+2`).
- `operador-ce`: Captura Esquerda (`l, c` salta para `l, c-2`).
- `operador-cc`: Captura Cima (`l, c` salta para `l-2, c`).
- `operador-cb`: Captura Baixo (`l, c` salta para `l+2, c`).

3.4. Heurística

São definidas duas heurísticas no ficheiro `puzzle.lisp` para uso com o algorítmo A*.

- **Heurística H1 (Base: Mobilidade)** (`h1-base`): \(h(x) = \frac{1}{\text{o}(x) + 1}\) Onde \(\text{o}(x)\) é o número de peças que conseguem mover-se (`contar-pecas-moveis`). **Privilegia tabuleiros com mais peças móveis**.
- **Heurística H2 (Extra: Contagem de Pinos)** (`h2-extra`): \(h(x) = \text{Número de Pinos} - 1\) Utiliza `contar-pinos` para obter o número total de pinos. **Estima a distância exata ao objetivo** (sem contar bloqueios).

---

4. Estruturas e Algoritmos de Procura (Módulo `procura.lisp`)

4.1. Tipo Abstrato de Dados: Nó

A estrutura do nó é uma lista que armazena os dados essenciais para os algoritmos de procura: \(\text{Nó} = (\text{estado } g \text{ } h \text{ pai})\).

| Seletor | Valor | Função |

| **Estado** | Estado do tabuleiro | `no-estado` | | **Custo \(g(n)\)** | Profundidade/Custo do caminho percorrido | `no-g` | | **Heurística \(h(n)\)** | Custo estimado até ao objetivo | `no-heuristica` | | **Pai** | Nó predecessor | `no-pai` | | **Custo Total \(f(n)\)** | \(g(n) + h(n)\) | `no-custo` |

4.2. Algoritmos de Procura

Os algoritmos de procura retornam uma lista com a seguinte estrutura: `(no-solucao, nós-gerados, nós-expandidos, tempo)`.

Procura em Largura Primeiro (BFS)

O BFS explora a árvore de procura nível a nível. A lista de abertos é tratada como uma fila (os sucessores são adicionados com `append` ao fim da lista).

Procura em Profundidade Primeiro (DFS)

O DFS explora a árvore de procura em profundidade. Permite limitar o caminho pela `prof-max`. A lista de abertos é tratada como uma pilha (os sucessores são adicionados no início da lista).

Algoritmo A* (Melhor Primeiro)

O A* utiliza a função de custo \(f(n) = g(n) + h(n)\) para guiar a pesquisa. A lista de **abertos é sempre ordenada** pela função `ordenar-nos` para priorizar o menor custo \(f(n)\). O algoritmo verifica a lista de `fechados` para evitar a re-expansão de nós se já foi encontrado um caminho de custo inferior ou igual.

4.3. Métricas de Desempenho

As métricas são calculadas para avaliar a eficiência dos algoritmos.

- **Penetrância** (`penetrancia L T-total`): $\text{Penetrância} = \frac{L}{T_{\text{total}}}$, onde L é o comprimento da solução e T_{total} é o número de nós gerados.
- **Ramificação Média** (`ramificacao-media L T-total`): Calculada através do método da Bissecção (`bisseccao L T-alvo min max erro`). O método resolve a equação do Polinómio B: $\text{Polinómio B} (b, L) = \begin{cases} 1 + L & \text{se } b = 1 \\ \frac{b^{L+1} - 1}{b - 1} & \text{se } b \neq 1 \end{cases}$

---

5. Interface e Visualização (Módulo `projeto.lisp`)

5.1. Fluxo de Utilização

O menu principal (`menu-principal`) permite selecionar a opção "Resolver um problema". O utilizador deve escolher o ID do problema e um dos quatro algoritmos de procura disponíveis.

5.2. Visualização da Solução

A função `mostrar-solucao` exibe o resultado, incluindo:

- O **Número de passos** da solução.
- **Estatísticas**: Nós Gerados, Expandidos e Tempo de execução.
- **Métricas**: Penetrância e Ramificação Média.
- O **estado do tabuleiro** (g e h) em cada passo do caminho.

5.3. Registo de Logs

Os resultados são gravados no ficheiro `log.dat` através da função `escrever-log`. O registo inclui a solução (em passos), nós gerados, nós expandidos, tempo, penetrância e ramificação média.