# **Projeto Nº 1**: Época Normal

Inteligência Artificial
Escola Superior de Tecnologia de Setúbal
2025/2026

Prof. Joaquim Filipe
Eng. Filipe Mariano

## 1. Descrição do Jogo - Solitário

O Solitário é um jogo de tabuleiro para um jogador composto por uma base em forma de cruz, geralmente com 33 cavidades, onde são colocados pinos (ou bolas) em todas as posições, excepto a central, que começa vazia. O objetivo é eliminar os pinos movendo um pino por cima de outro — apenas na horizontal ou vertical — até uma casa vazia imediatamente após. O pino que ficou entre os dois é então retirado do tabuleiro, como se tivesse sido "comido" nessa jogada. O jogo termina quando há apenas um pino no tabuleiro, exigindo raciocínio e planeamento para evitar bloqueios prematuros.

Figura 1: Tabuleiro real do Solitário.

O tabuleiro pode ser visto como uma matriz 7x7 (7 linhas e 7 colunas) onde ● = pino, ○ = casa vazia e (espaço) = posição inválida fora da cruz.

```
   1 2 3 4 5 6 7
1      ● ● ●
2      ● ● ●
3  ● ● ● ● ● ● ●
4  ● ● ● ○ ● ● ●
5  ● ● ● ● ● ● ●
6      ● ● ●
7      ● ● ●
```

Desta forma, pode se constatar que as seguintes posições fora da cruz são inválidas (`(x,y)` - coordenada da casa no tabuleiro em que `x` representa a linha e `y` a coluna):

- Linha 1 → (1,1), (1,2), (1,6), (1,7)
- Linha 2 → (2,1), (2,2), (2,6), (2,7)
- Linha 6 → (6,1), (6,2), (6,6), (6,7)
- Linha 7 → (7,1), (7,2), (7,6), (7,7)

## 1.1. Regras

O jogo desenrola-se da seguinte forma:

- O jogador move um pino na horizontal ou vertical, passando por cima de outro pino e ficando em uma casa vazia imediatamente após;
  - O pino que foi passado por cima é retirado do tabuleiro;
  - **Não é possível mover um pino na horizontal ou vertical sem que se passe por cima de outro pino**;
- O jogo continua até que não seja possível realizar nenhum movimento válido.

**Nota**: Têm aqui um exemplo em vídeo de como se desenrola o puzzle.

## 1.2. Exemplo de Jogada

Para exemplificar uma jogada no puzzle, podemos observar o tabuleiro como uma matriz 7x7 onde ● = pino, ○ = casa vazia e (espaço) = posição inválida fora da cruz. Na representação em baixo as linhas e colunas estão numeradas para facilitar a descrição da jogada.

#### Tabuleiro Inicial

O tabuleiro inicial (padrão) começa com a casa (4,4) vazia.

```
   1 2 3 4 5 6 7  ← colunas
1      ● ● ●
2      ● ● ●
3  ● ● ● ● ● ● ●
4  ● ● ● ○ ● ● ●
```

```
5  ● ● ● ● ● ● ●
6        ● ● ●
7        ● ● ●
```

**Jogada exemplo: mover o pino da posição (4,2) para (4,4)**

Passos:

1. **Escolher o pino que vai mover**: o pino em (4,2) (linha 4, coluna 2).
2. **Verificar a casa intermédia**: entre (4,2) e (4,4) está (4,3); tem lá um pino? Sim (●) — portanto é possível passar por cima.
3. **Verificar a casa de destino**: (4,4) é ○ (vazia) — pode receber o pino.
4. **Executar o movimento**:
    - O pino de (4,2) move-se para (4,4).
    - O pino que estava em (4,3) é retirado do tabuleiro.
    - A casa (4,2) fica vazia.

**Tabuleiro antes da jogada (com posições numeradas)**

```
   1 2 3 4 5 6 7
1        ● ● ●
2        ● ● ●
3  ● ● ● ● ● ● ●
4  ● ● ● ○ ● ● ●      ← aqui (4,2)=●, (4,3)=●, (4,4)=○
5  ● ● ● ● ● ● ●
6        ● ● ●
7        ● ● ●
```

**Tabuleiro após a jogada**

```
   1 2 3 4 5 6 7
1        ● ● ●
2        ● ● ●
3  ● ● ● ● ● ● ●
4  ● ○ ○ ● ● ● ●      ← agora (4,2)=○, (4,3)=○ (removido), (4,4)=●
5  ● ● ● ● ● ● ●
6        ● ● ●
7        ● ● ●
```

**Explicação**

O pino de (4,2) passou por cima do pino de (4,3) e ocupou a casa vazia (4,4), logo o pino que ficou no meio (4,3) foi então retirado do tabuleiro. Assim ficámos com uma peça a menos no tabuleiro.

### 1.3. Fim do Puzzle

O jogo termina quando **existir apenas um pino no tabuleiro**.

## 2. Objetivo do Projeto

No âmbito deste projeto o principal objetivo consiste em explorar o espaço de possibilidades tentando vários caminhos possíveis até encontrar a solução, recorrendo aos algoritmos de procura lecionados nas aulas. Neste sentido, o intuito do projeto passa por testar diversos tabuleiros com diversos algoritmos e analisar as soluções encontradas.

Pretende-se que os alunos desenvolvam um programa, em `Lisp`, aplicando diversos métodos de procura em espaço de estados conforme se indica detalhadamente mais adiante, para apresentar a sequência de estados ou de jogadas que conduzem de uma posição inicial do puzzle até uma posição final. A solução óptima consiste no caminho com menor número de jogadas entre o estado inicial e o estado final.

## 3. Formulação do Problema

### 3.1. Tabuleiro

O tabuleiro é representado sob a forma de uma lista de listas em lisp em que [nil] representa um espaço ilegal no tabuleiro, [0] representa uma casa vazia e [1] representa um lugar com peça. Cada uma dessas listas é composta por 7 elementos, cujo valor de cada casa está supramencionado.

```
(
 (nil nil 1 1 1 nil nil)
 (nil nil 1 1 1 nil nil)
 ( 1   1  1 1 1  1   1)
 ( 1   1  1 0 1  1   1)
 ( 1   1  1 1 1  1   1)
 (nil nil 1 1 1 nil nil)
 (nil nil 1 1 1 nil nil)
)
```

### 3.2. Operadores

Para este problema a estrutura de cada movimento pode ser representado da seguinte forma:

```
(TipoDeMovimento PosiçãoHorizontal PosiçãoVertical)
```

- **PosiçãoHorizontal** e **PosiçãoVertical** indicam as coordenadas da casa de origem do pino a ser movido.

- **TipoDeMovimento** define a direção e o tipo de ação de captura, visto que se pode movimentar na horizontal ou vertical.

| Código | Significado | Descrição |
|---|---|---|
| **cd** | Captura Direita | O pino move-se duas casas à direita, passando por cima de outro pino. |
| **ce** | Captura Esquerda | O pino move-se duas casas à esquerda, passando por cima de outro pino. |
| **cc** | Captura Cima | O pino move-se duas casas para cima, passando por cima de outro pino. |
| **cb** | Captura Baixo | O pino move-se duas casas para baixo, passando por cima de outro pino. |

**Nota**: Apenas as capturas (cd, ce, cc, cb) são movimentos válidos, pois o pino só se pode mover saltando por cima de outro.

#### *3.2.1. Captura Direita — cd*

Exemplo: **(cd 4 2)**
Descrição: O pino da posição (4,2) move-se duas casas à direita, saltando sobre o pino em (4,3) e ocupando a casa (4,4).

**Antes:**

```
Linha 4: ● ● ● ○ ● ● ●
           ↑ ↑
           | |
         (4,2)(4,3)
```

**Depois:**

```
Linha 4: ● ○ ○ ● ● ● ●
```

#### *3.2.2. Captura Esquerda — ce*

Exemplo: **(ce 4 6)**
Descrição: O pino da posição (4,6) move-se duas casas à esquerda, saltando sobre o pino em (4,5) e ocupando a casa (4,4).

**Antes:**

```
Linha 4: ● ● ● ○ ● ● ●
                ↑ ↑
                | |
              (4,5)(4,6)
```

**Depois:**

```
Linha 4: ● ● ● ● ○ ○ ●
```

### *3.2.3. Captura Cima — cc*

Exemplo: **(cc 6 4)**
Descrição: O pino da posição (6,4) move-se duas casas para cima, saltando sobre o pino em (5,4) e ocupando a casa (4,4).

**Antes:**

```
Coluna 4:
 ●
 ●
 ●
 ○
 ●
 ●
 ●
```

**Depois:**

```
Coluna 4:
 ●
 ●
 ●
 ●
 ○
 ○
 ●
```

### *3.2.4. Captura Baixo — cb*

Exemplo: **(cb 2 4)**
Descrição: O pino da posição (2,4) move-se duas casas para baixo, saltando sobre o pino em (3,4) e ocupando a casa (4,4).

**Antes:**

```
Coluna 4:
 ●
 ●
 ●
```

- ○
- ●
- ●
- ●

**Depois:**

```
Coluna 4:
```
- ●
- ○
- ○
- ●
- ●
- ●
- ●

## 3.3. Solução Encontrada

A solução pode representar-se por uma sequência de estados, desde o estado inicial até ao estado final. **A solução pode representar-se por uma sequência de estados, desde o estado inicial até ao estado final, ou por uma sequência de operações realizadas sobre os pinos do tabuleiro, indicando o tipo de movimento seguido da posição da peça a ser movida (onde está a peça antes de iniciar o movimento).**

## 3.4. Estrutura do Programa

O programa deverá estar dividido em três partes, cada uma num ficheiro diferente:

1. Uma parte com a implementação dos métodos de procura, de forma independente do domínio de aplicação;
2. Outra para implementar a resolução do problema, incluindo a definição dos operadores e heurísticas, específicos do domínio de aplicação;
3. E a terceira parte para fazer a interação com o utilizador e para proceder à escrita e leitura de ficheiros.

Enquanto a primeira parte do programa deverá ser genérica para qualquer problema que possa ser resolvido com base no método de procura selecionado, a segunda parte é específico do domínio de aplicação, nomeadamente o problema que este projeto se propõe resolver.

O projeto deverá apresentar um estudo comparativo do comportamento dos três métodos seguintes, conforme explicado na secção 4: procura em largura (BFS), procura em profundidade (DFS) e A*.

Para além destas três formas de procura, cada grupo pode programar, aplicar e estudar os algoritmos Simplified Memory A* (SMA*), Iterative Deepening A* (IDA*) e Recursive BestFirst Search (RBFS), que representarão um bónus para quem os implementar (ver Secção 9).

No caso dos métodos informados, o programa deverá utilizar funções heurísticas modulares, ou seja, que possam ser colocadas ou retiradas do programa de procura como módulos.

As heurísticas não devem estar embutidas de forma rígida no programa de procura. Exige−se a utilização de duas heurísticas, uma fornecida no fim do presente documento e outra desenvolvida pelos alunos.

O projeto deverá incluir a implementação de cada um dos métodos, de forma modular, permitindo que o utilizador escolha qualquer um deles, conjuntamente com os seus parâmetros (heurística, profundidade, etc.) para a resolução de um dado problema.

## 4. Experiências e Estudos

Pretende-se que o projeto estude, para cada problema fornecido em anexo, o desempenho de cada algoritmo e, no caso dos algoritmos de procura informados, de cada uma das heurísticas propostas, apresentando, em relação a cada problema:

- A solução encontrada;
- Dados estatísticos sobre a sua eficiência, nomeadamente:
    - Fator de ramificação médio
    - Número de nós gerados
    - Número de nós expandidos
    - Penetrância
    - Tempo de execução
    - Caminho até à solução.

Os projetos deverão apresentar os dados acima referidos num ficheiro produzido automaticamente pelo programa, sendo descontado `0,5` valor por cada problema sem os resultados apresentados. No caso de ser apresentada a solução, mas não o estudo de desempenho o desconto é de apenas `0,2` valor por cada caso.

Estes problemas deverão estar num ficheiro `problemas.dat`, utilizando a notação atrás indicada. O último problema (`G`) será apresentado durante a avaliação oral e inserido no ficheiro `problemas.dat` para verificar o funcionamento do projeto.

## 5. Heurísticas

Sugere-se usar como heurística de base, uma heurística que privilegia os tabuleiros com o maior número de peças capazes de serem movidas. Para um determinado tabuleiro x:

h(x) = 1 / ( o(x) + 1 )

em que:

- o(x) é o número de peças possíveis de serem movidas.

Esta heurística pode ser melhorada para refletir de forma mais adequada o conhecimento acerca do puzzle e assim contribuir para uma maior eficiência dos algoritmos de procura informados.

Além da heurística acima sugerida, deve ser definida pelo menos uma segunda heurística que deverá melhorar o desempenho dos algoritmos de procura informados em relação à primeira fornecida.

## 6. Grupos

Os projetos deverão ser realizados em grupos de, no máximo, três pessoas sendo contudo sempre sujeitos a avaliação oral individual para confirmação da capacidade de compreensão dos algoritmos e de desenvolvimento de código em LISP.

O grupo poderá ser constituido por alunos que frequentaram turmas diferentes.

# 7. Datas

**Entrega do projeto:** 12 de Dezembro de 2025, até as 23:59.

# 8. Documentação a Entregar

A entrega do projeto e da respetiva documentação deverá ser feita através do Moodle, na zona do evento "Entrega do Projeto em Época Especial". Todos os ficheiros a entregar deverão ser devidamente arquivados num ficheiro comprimido (`ZIP` com um tamanho máximo de 5Mb), até à data acima indicada. O nome do arquivo deve seguir a estrutura `<iniciaisAluno1>_<numeroAluno1>_<iniciaisAluno2>_<numeroAluno2>_<iniciaisAluno3>_<numeroAluno3>_P1`.

## 8.1. Código fonte

Os ficheiros de código devem ser devidamente comentados e organizados da seguinte forma:

**projeto.lisp** Carrega os outros ficheiros de código, escreve e lê ficheiros, e trata da interação com o utilizador.

**puzzle.lisp** Código relacionado com o problema.

**procura.lisp** Deve conter a implementação de:

1. Algoritmo de Procura de Largura Primeiro (BFS)
2. Algoritmo de Procura do Profundidade Primeiro (DFS)
3. Algoritmo de Procura do Melhor Primeiro (A*)
4. Os algoritmos SMA*, IDA* e/ou RBFS (caso optem por implementar o bónus)

## 8.2. Problemas

Deverá haver um ficheiro de problemas, com a designação `problemas.dat`, contendo todos os exemplos de tabuleiro que se quiser fornecer ao utilizador, organizados de forma sequencial, e que este deverá escolher mediante um número inserido no interface com o utilizador.

Esse número representa o número de ordem na sequência de exemplos. O ficheiro deverá ter várias listas, separadas umas das outras por um separador legal, e não uma lista de listas. Essas listas serão tantas quantos os problemas fornecidos.

Na oral, os docentes irão solicitar que se adicione mais um exemplo, numa dada posição do ficheiro, que deverá imediatamente passar a ser selecionável através do interface com o utilizador e ser resolvido normalmente.

## 8.3. Manuais

No âmbito da Unidade Curricular de Inteligência Artificial pretende–se que os alunos pratiquem a escrita de documentos recorrendo à linguagem de marcação ***Markdown***, que é amplamente utilizada para os ficheiros ***ReadMe*** no ***GitHub***. Na Secção 4 do guia de Laboratório nº 2, encontrará toda a informação relativa à estrutura recomendada e sugestões de ferramentas de edição para ***Markdown***.

Para além de entregar os ficheiros de código e de problemas, é necessário elaborar e entregar 2 manuais (o manual de utilizador e o manual técnico), em formato PDF juntamente com os sources em MD, incluídos no arquivo acima referido.

**Manual Técnico:**

O Manual Técnico deverá conter o algoritmo geral, por partes e devidamente comentado; descrição dos objetos que compõem o projeto, incluindo dados e procedimentos; identificação das limitações e opções técnicas. Deverá ser apresentada uma análise critica dos resultados das execuções do programa, onde deverá transparecer a compreensão das limitações do projeto. Deverão usar uma análise comparativa do conjunto de execuções do programa para cada algoritmo e cada problema, permitindo verificar o desempenho de cada algoritmo e das heurísticas. Deverá, por fim, apresentar a lista dos requisitos do projeto (listados neste documento) que não foram implementados.

**Manual de Utilizador:**

O Manual do Utilizador deverá conter a identificação dos objetivos do programa, juntamente com descrição geral do seu funcionamento; explicação da forma como se usa o programa (acompanhada de exemplos); descrição da informação necessária e da informação produzida (ecrã/teclado e ficheiros); limitações do programa (do ponto de vista do utilizador, de natureza não técnica).

# 9. Avaliação

Tabela 1: Grelha de classificação.

| **Funcionalidade** | **Valores** |
|---|---|
| Representação do estado e operadores | 2.5 |
| Funções referentes ao puzzle | 2.5 |
| Procura em profundidade e largura | 2.5 |
| Procura com A* e heurística dada | 2.5 |
| Implementação de nova heurística | 1.5 |
| Resolução dos problemas a) a f) | 3 |
| Resolução do problema g) (introduzido no ficheiro de problemas após a entrega) | 0.5 |
| Qualidade do código | 2 |
| Interação com o utilizador | 1 |
| Manuais (utilizador e técnico) | 2 |
| **Total** | 20 |
| **Bónus** | 3 |

O valor máximo da nota são 20 valores, já com a integração do valor do bónus.

Os problemas a) a f) estão descritos na Secção Experiências, enquanto que o problema g) será adicionado posteriormente ao ficheiro de problemas. A avaliação do projeto levará em linha de conta os seguintes

aspectos:

- Data de entrega final – Existe uma tolerância de 3 dias em relação ao prazo de entrega, com a penalização de 1 valor por cada dia de atraso. Findo este período a nota do projeto será 0.
- Correção processual da entrega do projeto – (Moodle; manuais no formato correto). Anomalias processuais darão origem a uma penalização que pode ir até 3 valores.
- Qualidade técnica – Objetivos atingidos; Código correto; Facilidade de leitura e manutenção do programa; Opções técnicas corretas.
- Qualidade da documentação – Estrutura e conteúdo dos manuais que acompanham o projeto.
- Avaliação oral – Eficácia e eficiência da exposição; Compreensão das limitações e possibilidades de desenvolvimento do programa. Nesta fase poderá haver lugar a uma revisão total da nota de projeto.

# 10. Recomendações Finais

Com este projeto pretende–se motivar o paradigma de programação funcional. A utilização de variáveis globais, de instruções de atribuição do tipo set, setq, setf (somente dentro de *closures*), de ciclos, de funções destrutivas ou de quaisquer funções com efeitos laterais é fortemente desincentivada dado que denota normalmente uma baixa qualidade técnica. A sequenciação só será permitida no contexto das funções de leitura/escrita.

As únicas excepções permitidas a estas regras poderão ser a utilização da instrução loop para implementar funções de escrita ou leitura em ficheiros.

A utilização de ferramentas como **ChatGPT**, **Copilot** ou outras **é permitida**, desde que apenas como auxílio, e não para gerar o código completo do projeto. Todo o código produzido com a ajuda destas ferramentas deve conter um comentário indicando que foi gerado automaticamente. Grupos que não incluam esta indicação ou em que existam suspeitas de código integralmente produzido por ferramentas de IA poderão ser penalizados na nota final ou até ter o projeto anulado.

**ATENÇÃO:** Suspeitas confirmadas de plágio serão penalizadas com a anulação de ambos os projetos envolvidos (fonte e destino), e os responsáveis ficam sujeitos à instauração de processo disciplinar.

# Anexos - Problemas

Problema A

```
   1 2 3 4 5 6 7
1      o o o
2      o o o
3  o o o o o o o
4  o o o o o o o
5  o o o o ● ● o
6      o ● o
7      o o o
```

Problema B

```
   1 2 3 4 5 6 7
1        ○ ○ ○
2        ○ ○ ○
3  ○ ○ ○ ○ ○ ○ ○
4  ○ ○ ○ ● ○ ○ ○
5  ○ ○ ○ ● ● ● ○
6        ○ ○ ○
7        ○ ○ ○
```

---

Problema C

```
   1 2 3 4 5 6 7
1        ○ ○ ○
2        ○ ○ ○
3  ○ ○ ○ ○ ● ○ ○
4  ○ ○ ○ ○ ● ● ○
5  ○ ○ ○ ● ● ● ○
6        ○ ○ ○
7        ○ ○ ○
```

---

Problema D

```
   1 2 3 4 5 6 7
1        ○ ○ ○
2        ○ ○ ●
3  ○ ○ ○ ○ ● ● ○
4  ○ ○ ○ ○ ● ● ●
5  ○ ○ ○ ● ● ● ●
6        ○ ○ ○
7        ○ ○ ○
```

---

Problema E

```
   1 2 3 4 5 6 7
1        ○ ○ ○
2        ○ ○ ○
3  ○ ○ ○ ● ● ● ●
4  ○ ○ ○ ○ ● ● ●
5  ○ ○ ○ ● ● ● ●
```

```
6      o o o
7      o o o
```

---

Problema F

```
   1 2 3 4 5 6 7
1      ● ● ●
2      ● ● ●
3  ● ● ● ● ● ● ●
4  ● ● ● o ● ● ●
5  ● ● ● ● ● ● ●
6      ● ● ●
7      ● ● ●
```