Manual Técnico - Projeto Jogo do Solitário (Fases 1 e 2)

Inteligência Artificial 2025/26

Docentes: Prof. Joaquim Filipe e Eng. Filipe Mariano

Data: 16 de janeiro de 2026

Aluno	Número
António Guerreiro	202200160
Bruno Leite	202100504
Guilherme Cruz	2024129841

---

1. Introdução

O projeto evoluiu de um sistema de procura em espaço de estados (Fase 1) para um motor de jogo adversarial (Fase 2). O objetivo da Fase 2 é permitir que dois jogadores (Humano ou CPU) compilam num tabuleiro de Solitário 2, onde a vitória é alcançada ao atingir a base inicial do adversário ou através de vantavem material.

---

2. Arquitetura do Sistema

A arquitetura foi modularizada em cinco ficheiros de código e um de dados para garantir a separação entre o motor de jogo e o domínio:

Ficheiro	Conteúdo Principal
`projeto.lisp`	Gestão de menus, integração de fases e carregamento de ficheiros.
`puzzle.lisp`	Domínio do problema: operadores de movimento (simples e captura), representação do tabuleiro e funções de avaliação.
`procura.lisp`	Implementação genérica do algoritmo **Negamax com cortes Alfa-Beta**.
`jogo.lips`	Lógica de jogo adversarial, geração de sucessores para 2 jogadores e interface de campeonato.
`problemas.dat`	Estados iniciais dos tabuleiros.

---

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

3.1. Representação do Tabuleiro

O tabuleiro 7x7 é representado como uma **lista de listas**.

- **1/2:** Peças do Jogador 1 e Jogador 2, respetivamente.
- **0:** Cavidade vazia.
- **nil:** Posições fora da geometria em cruz.

3.2. Operadores de Transição

Foram implementados 8 operadores funcionais que garantem a imutabilidade do estado (geram sempre um novo tabuleiro):

- **Movimentos Simples:** `d, e, c, b` (deslocação para casa adjacente vazia).
- **Movimentos de Captura:** `cd, ce, cc, cb` (salto sobre peça do adversário).

---

4. Algoritmos de Decisão (Módulo `algoritmo.lisp`)

4.1. Negamax com Cortes Alfa-Beta

Para a Fase 2, foi implementado o algoritmo **Negamax**, uma variante simplificada do Minimax para jogos de soma zero.

- **Cortes Alfa-Beta:** Otimizam a exploração da árvore de jogo, podando ramos que não alterariam a decisão final, reduzindo o fator de ramificação efetivo.
- **Gestão de Tempo:** O algoritmo monitoriza o tempo de execução através de `get-internal-real-time`, forçando o retorno da melhor jogada encontrada caso o limite definido (1 a 20) seja atingido.
- **Otimização de Cauda (TCO):** As funções utilizam declarações `(declare (optimize (speed 3)))` para garantir que a recursividade seja otimizada pelo compilador, prevenidos erros de *stack overflow*.

---

5. Funções de Avaliação e Heurísticas

5.1. Heurísticas de Procura (Fase 1)

- **H1 (Mobilidade):** Favorece estados com maior número de peças capazes de se mover.
- **H2 (Contagem):** Estimativa baseada no número de pinos restantes.

5.2. Função de Avaliação Adversarial (Fase 2)

A função `avaliar-estado` utiliza uma combinação de:

1. **Vantagem Material:** Diferença entre o número de pinos próprios e do adversário.
2. **Posicionamento:** Bónus por ocupação de casas estratégicas e proximidade às linhas de base do oponente.

---

6. Métricas de Desempenho (Módulo `procura.lisp`)

O sistema calcula métricas rigorosas para análise da eficiência:

- **Penetrância:** Razão entre a profundidade da solução e o total de nós gerados.
- **Fator de Ramificação Médio:** Calculado recursivamente através do método da bissecção sobre o polinómio de ramificação:

$$B(b, L) = \frac{b^{L+1} - 1}{b - 1}$$

---

7. Notas de Implementação Técnica

- **Geração de Sucessores:** Otimizada em `jogo.lisp` recorrendo à função `mapcan` em vez de `append` recursivo, reduzindo drasticamente o consumo de memória.
- **Regras de Abertura:** Implementação através de um filtro que restringe os movimentos iniciais (J1 apenas 'b' na linha 2; J2 apenas 'c' linha 6).