Implementação Lógica do jogo 'Quem Quer Ser Milionário'

Lógica e Inteligência Artificial

14.01.2025

Pedro Reis PG59908

João Azevedo PG61693

Guilherme Pinto PG60225

Luís Silva PG60390

Diogo Azevedo PG61217

1 . Introdução

Contextualização e Paradigma

Este projeto desenvolve um agente de **IA Simbólica** em *Prolog* que recria o concurso *"Quem Quer Ser Milionário"*, substituindo a execução imperativa pela dedução lógica.

Mudança de Paradigma:

- **Imperativo:** Define a sequência de passos para mudar o estado.
- **Declarativo:** Define factos e relações para que o motor descubra o caminho.

Motivação e Desafios

A escolha do *Prolog* visa alinhar a teoria da **Lógica Formal** com a prática de engenharia de software, enfrentando as limitações da imutabilidade.

Motivação:

- **Modelação:** Estruturar conhecimento hierárquico em vez de dados lineares.
- **Formalismo:** Substituir o `if/else` por *Modus Ponens* e *Modus Tollens*.

Principais Desafios:

- **Gestão de Estado:** Sem variáveis globais, o estado do jogo persiste apenas via **recursividade**.
- **Multimédia:** Sincronizar áudio e animações num ambiente "single-threaded" lógico.

Objetivos do Projeto

O desenvolvimento foca-se na integridade das regras e na experiência de utilização, estruturado em 5 eixos fundamentais:

1. **Base de Conhecimento:** Estruturação hierárquica eficiente.
2. **Motor de Inferência:** Ciclo de jogo recursivo para gestão de transição de estados.
3. **Mecânica de Ajudas:** Algoritmos de manipulação de listas.
4. **Regras Formais:** Aplicação explícita de lógica para controlo narrativo.
5. **Interface:** Motor de renderização para terminal (ANSI/ASCII) com tratamento de erros.

2 . Estrutura Lógica e Arquitetura

Mapa da Arquitetura do Sistema

O projeto organiza-se em módulos especializados. Abaixo apresenta-se a distribuição dos principais ficheiros:

Categoria	Descrição	Ficheiros
Persistência	Base de conhecimento estática e histórico	`perguntas.pl` `ranking.pl`
Motor Lógico	Controlo de fluxo e regras de inferência	`milionario.pl` `regras.pl`
Interface	Renderização visual e input	`estudio.pl` `menu.pl`
Multimédia	Gestão de áudio e motor de vídeo	`audio.pl` `animacao.pl`
Integração	Ponte para Python e LLMs	`telefonar.pl` `ask_ollama.py`

Persistência e Dados (`perguntas.pl` e `ranking.pl`)

Nesta camada definemos as estruturas de dados estáticas e a memória do sistema.

Base de Conhecimento (`perguntas.pl`):

- Define-se o predicado atómico `pergunta/6`, que encapsula ID, enunciado, opções, resposta certa, nível e categoria.
- **Opções:** Aqui optamos por estruturar as opções em **lista**. Isto permite a permutação aleatória em tempo de execução sem alterar a integridade lógica da resposta correta.

Histórico (`ranking.pl`):

- Define-se a persistência através da escrita direta de **termos Prolog** em ficheiro.
- Elimina-se a necessidade de *parsers* (como JSON/CSV), já que os dados são lidos nativamente como código executável.

Motor de Inferência (`milionario.pl` e `regras.pl`)

Aqui define-se o controlo de fluxo e a validação lógica do jogo.

Ciclo de Jogo (`milionario.pl`):

- Define-se o predicado recursivo `ciclo_jogo/10`.
- Este predicado gere a máquina de estados (pontuação, vidas, ajudas) através da passagem de argumentos no *stack*, contornando a imutabilidade das variáveis em Prolog.
- Aplica-se formalmente o **Modus Ponens** e o **Modus Tollens**.

Normas (`regras.pl`):

- Define-se a separação entre "lógica de execução" e "parâmetros de jogo".
- Estabelecem-se os factos estáticos para os **Checkpoints** e as condições de vitória/derrota.

Interface e Visualização (`estudio.pl` e `menu.pl`)

Nesta camada definem-se os mecanismos de interação com o terminal.

Renderização (`estudio.pl`):

- Define-se um **pipeline** sequencial de apresentação: Limpeza → Background → Texto.
- Centraliza-se a manipulação de códigos ANSI para posicionamento de cursor e cores.

Tratamento de Input (`menu.pl`):

- Define-se uma máquina de estados para leitura de teclado.
- Implementa-se a lógica de deteção de teclas especiais (*Backspace*, *Enter*) para garantir que a correção visual do input ocorre antes do processamento lógico.

Multimédia e Otimização (`audio.pl` e `animacao.pl`)

Aqui definem-se os processos para contornar a execução *single-threaded* do Prolog.

Áudio (`audio.pl`):

- Define-se a criação de **Threads** destacadas para a execução de som.
- Estabelece-se uma ponte com o **PowerShell** para delegar a reprodução de áudio ao sistema operativo.

Vídeo (`animacao.pl`):

- Define-se a otimização de I/O através de transferência binária.
- Implementa-se lógica de **Frame Skipping**: o motor calcula matematicamente se deve saltar frames para manter a sincronização temporal em máquinas mais lentas.

Integração Híbrida (`telefonar.pl` e `ask_ollama.py`)

Nesta camada define-se a comunicação entre Lógica Simbólica e Probabilística.

Interface (`telefonar.pl`):

- Define-se a criação de processos externos para invocar o interpretador Python.
- Gere-se a comunicação via **Pipes** com codificação UTF-8 forçada.

Script de Inferência (`ask_ollama.py`):

- Define-se a interação com o LLM.
- Aplica-se sanitização via **Regex** e **Prompt Engineering** restritivo para converter a saída textual do modelo num único caractere (A-D) compreensível pelo Prolog.

3 . Demonstração e Criatividade

Estética Visual e Inspiração

Para dar "vida" ao terminal, criámos uma estética inspirada nos **sprites** de treinadores de **Pokémon**. O motor converteu imagens reais em **pixel art** ANSI, adaptando a resolução sem perder a identidade.

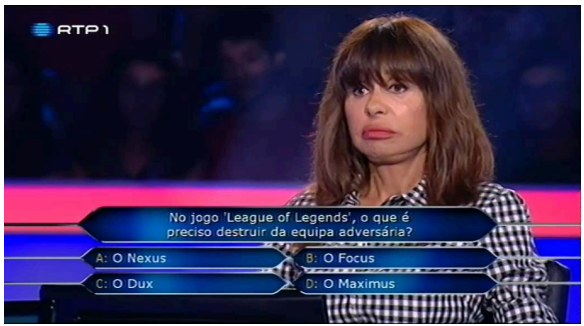

Figure 1: Original

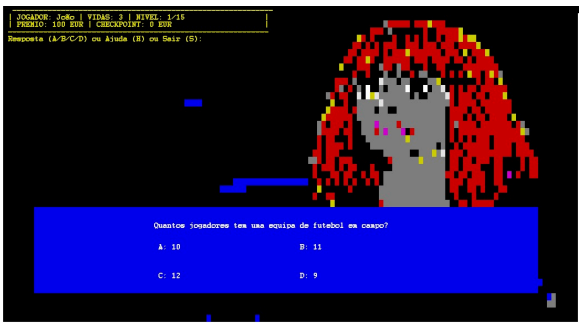

Figure 2: Resultado no Jogo

4 . Conclusão