Universidade do Minho
Mestrado em Inteligência Artificial

Unidade curricular de
Lógica e Inteligência Artificial
Ano Lectivo 2025/2026

MIAlionário: Desafiando a Base de Conhecimento através da Programação Lógica

Implementação do Jogo “Quem Quer Ser Milionário” em Prolog

Carlos Bergueira	PG 11605
Diego Silva	PG 59999
Filipa Pereira	PG 42201
Rui Rodrigues	PG 7942

Janeiro, 2026

MIAlionário: Desafiando a Base de Conhecimento através da Programação Lógica

Implementação do Jogo “Quem Quer Ser Milionário” em Prolog

Carlos Bergueira	PG 11605
Diego Silva	PG 59999
Filipa Pereira	PG 42201
Rui Rodrigues	PG 7942

Janeiro, 2026

Resumo

Este projeto apresenta o “MIAlionário”, uma implementação digital e inteligente do icónico jogo “Quem Quer Ser Milionário”, desenvolvida no âmbito do Mestrado em Inteligência Artificial (MIA). Utilizando a linguagem de programação lógica Prolog como motor central, o sistema combina uma base de conhecimento rigorosa com uma interface *web* moderna. O sistema utiliza o SWI-Prolog para gerir o motor de inferência lógica e uma API REST para comunicar com um frontend em JavaScript, garantindo uma separação clara entre a base de conhecimento e a camada de apresentação.

Pontos Principais:

- Fundamentação Lógica: Aplicação de regras de inferência formal, como *Modus Ponens* e *Modus Tollens*, para determinar o progresso do jogador com base nas suas respostas.
- Arquitetura do Sistema: Utilização de uma base de conhecimento modular (perguntas categorizadas por dificuldade), predicados dinâmicos para gestão de estado e um servidor HTTP para comunicação via JSON.
- Interface e Experiência: Implementação de dois modos de jogo — consola (textual) e *web* (gráfico) - incluindo funcionalidades clássicas como patamares de segurança e ajudas (*lifelines*) geridas no frontend.
- Objetivo Académico: Demonstrar a versatilidade e a viabilidade do Prolog em arquiteturas cliente-servidor contemporâneas, unindo a programação declarativa a tecnologias *web* atuais.

Índice

1. Introdução

O presente trabalho tem como objetivo demonstrar a aplicação prática da programação lógica em Prolog, recorrendo ao desenvolvimento de uma versão do jogo “Quem Quer Ser Milionário”. O projeto integra conceitos fundamentais de lógica formal, inferência lógica, programação declarativa e ainda a criação de uma interface de utilizador em modo consola e modo web, através de um servidor HTTP.

Este trabalho foi desenvolvido utilizando SWI-Prolog, explorando tanto as capacidades clássicas da linguagem como funcionalidades modernas, tais como serviços *web* e comunicação em formato JSON.

2. Objetivos do Projeto

Os principais objetivos deste projeto são:

- Aplicar programação lógica na resolução de um problema real
- Implementar regras de inferência lógica (*Modus Ponens* e *Modus Tollens*)
- Utilizar factos e predicados dinâmicos
- Desenvolver um jogo interativo em modo consola
- Criar uma API REST em Prolog
- Integrar um *frontend web com backend* em Prolog

3. Estrutura Geral do Sistema

O sistema encontra-se dividido em vários componentes:

- Base de conhecimento: perguntas do jogo
- Motor lógico: inferência baseada no estado do jogador
- Interface de consola: interação textual com o utilizador
- Servidor HTTP: comunicação com o *frontend web*
- API REST: *endpoints* para perguntas e validação de respostas

4. Importação de Módulos

O código recorre a vários módulos da biblioteca padrão do SWI-Prolog:

```
:- use_module(perguntas).
:- use_module(library(random)).
:- use_module(library(lists)).
```

Nos quais:

- O módulo *perguntas* contém a base de conhecimento com os factos pergunta/6 (pergunta(Id, Texto, Opções (a, b, c, d), RespostaCorreta, Nivel, Valor)
- O módulo *random* permite a geração aleatória de perguntas
- O módulo *lists* disponibiliza predicados auxiliares para manipulação de listas

Adicionalmente, são importados módulos HTTP que permitem a criação de um servidor *web* e a troca de dados em formato JSON.

5. Configuração do Servidor HTTP

O servidor HTTP é configurado através dos seguintes módulos:

:- use_module(library(http/thread_httpd)).

:- use_module(library(http/http_dispatch)).

:- use_module(library(http/http_parameters)).

:- use_module(library(http/http_json)).

O sistema suporta *CORS (Cross-Origin Resource Sharing)*, permitindo que o *frontend web* comunique com a API independentemente da origem.

6. Estado Lógico e Factos Dinâmicos

:- dynamic jogador_acertou/0.

:- dynamic jogador_errou/0.

Estes predicados representam o estado do jogador e são modificados dinamicamente durante a execução do jogo, permitindo a aplicação de inferência lógica.

7. Inferência Lógica

A inferência lógica é baseada na seguinte regra:

implica(resposta_certa, progresso).

7.1 *Modus Ponens*

Se o jogador acertar a resposta, o sistema conclui que houve progresso:

conclusao(progresso).

7.2 *Modus Tollens*

Caso o jogador erre a resposta, o sistema conclui que não houve progresso:

conclusao(nao_resposta_certa).

É também incluída uma falácia lógica intencional, com fins académicos, permitindo a análise crítica do raciocínio lógico.

8. Patamares de Segurança

O jogo define patamares mínimos garantidos:

patamar(5, 1000).
patamar(10, 5000).

Estes patamares asseguram que o jogador não perde totalmente o prémio após atingir determinados níveis.

9. Geração e Progressão de Perguntas

As perguntas encontram-se organizadas por níveis de dificuldade, sendo apresentadas de forma progressiva. Dentro de cada nível, a ordem das perguntas é aleatória, garantindo variedade entre execuções do jogo.

10. Interface em Consola

O modo consola apresenta:

- Perguntas e opções
- *Feedback* imediato ao jogador
- Cores ANSI para melhor experiência visual
- Reprodução de sons consoante os eventos do jogo

11. API REST

O sistema disponibiliza dois *endpoints* principais:

11.1 /api/question

Responsável por fornecer uma nova pergunta ao *frontend*, respeitando a progressão de níveis e evitando perguntas repetidas.

11.2 /api/check

Responsável por validar a resposta do jogador e devolver o resultado em formato JSON.

12. Inicialização Automática

O servidor HTTP é iniciado automaticamente aquando do carregamento do ficheiro Prolog:
:- initialization(iniciar_servidor).

13. Justificação Detalhada das Opções de Implementação

Nesta secção é apresentada a justificação académica e técnica das principais decisões tomadas ao longo do desenvolvimento do código Prolog.

13.1 Utilização de Módulos

A utilização extensiva de módulos (*use_module/1*) tem como principal objetivo a modularidade, reutilização de código e clareza estrutural. O módulo perguntas separa a base de conhecimento do restante sistema, respeitando o princípio da separação de responsabilidades. Os módulos random e lists são essenciais para operações não determinísticas e manipulação eficiente de listas.

Os módulos HTTP do SWI-Prolog foram escolhidos por permitirem a criação de um servidor *web* completo sem recorrer a tecnologias externas, demonstrando a capacidade do Prolog para aplicações modernas.

13.2 Factos Dinâmicos e Estado do Sistema

Os predicados dinâmicos *jogador_acertou/0 e jogador_errou/0* representam o estado lógico do jogo. Esta abordagem foi escolhida por refletir diretamente o paradigma declarativo do Prolog, onde o conhecimento pode ser atualizado dinamicamente à medida que novas informações se tornam verdadeiras ou falsas.

A utilização de *assertz/1 e retractall/1* permite simular memória de curto prazo, necessária para a inferência lógica entre perguntas consecutivas.

13.3 Inferência Lógica

A regra *implica(resposta_certa, progresso)* foi introduzida para modelar formalmente o raciocínio lógico subjacente ao jogo. A aplicação do *Modus Ponens* permite concluir progresso quando uma resposta correta é registada, enquanto o *Modus Tollens* é utilizado quando ocorre uma resposta errada.

A inclusão de uma falácia lógica intencional tem um propósito exclusivamente académico, permitindo discutir limitações e erros comuns em sistemas de inferência.

13.4 Progressão e Patamares de Segurança

Os patamares de segurança foram implementados para refletir as regras clássicas do jogo original. A decisão de os representar como factos *(patamar/2)* permite fácil extensão ou modificação das regras, mantendo a flexibilidade do sistema.

O predicado *atualiza_patamar/4* garante que o jogador mantém sempre o maior patamar atingido, mesmo após respostas incorretas.

13.5 Geração de Perguntas

A separação das perguntas por níveis de dificuldade justifica-se pela necessidade de uma progressão controlada. A utilização de *random_permutation/2* garante variabilidade sem comprometer a dificuldade esperada em cada fase do jogo.

Esta abordagem combina controlo determinístico (nível) com comportamento não determinístico (ordem das perguntas).

13.6 Interface em Consola

A interface em consola foi implementada com fins pedagógicos, permitindo observar diretamente a execução do programa Prolog. O uso de cores ANSI e sons tem como objetivo melhorar a experiência do utilizador e demonstrar integração com o sistema operativo.

13.7 Servidor HTTP e API REST

A implementação de um servidor HTTP em Prolog demonstra a versatilidade da linguagem. Os *endpoints /api/question* e */api/check* seguem princípios REST, utilizando JSON como formato de troca de dados, o que facilita a integração com um *frontend web.*

A ativação de *CORS* foi necessária para permitir comunicação entre diferentes origens, uma exigência comum em aplicações *web* modernas.

13.8 Inicialização Automática

A utilização do predicado *initialization/1* permite que o servidor seja iniciado automaticamente, simplificando a execução do sistema e reduzindo a necessidade de intervenção manual.

14. Justificação do Código JavaScript e Integração com Prolog

Esta secção apresenta a justificação detalhada do código JavaScript responsável pela interface *web* do jogo e pela comunicação com o *backend* desenvolvido em Prolog. O ficheiro analisado corresponde à página jogar.html, que contém estrutura HTML, estilos CSS e lógica JavaScript.

14.1 Separação de Responsabilidades

O sistema foi concebido segundo uma arquitetura cliente–servidor. O JavaScript é responsável exclusivamente pela interação com o utilizador (interface gráfica, temporizador, ajudas e *feedback* visual), enquanto o Prolog assume o papel de motor lógico, validação de respostas e gestão das perguntas. Esta separação garante modularidade, clareza e facilidade de manutenção.

14.2 Comunicação com o *Backend* Prolog

A comunicação entre o *frontend* e o *backend* é realizada através da API REST disponibilizada pelo servidor Prolog, recorrendo à função fetch. São utilizados dois endpoints principais:

- */api/question*: obtenção de novas perguntas
- */api/check*: verificação das respostas do jogador

Esta abordagem justifica-se por seguir práticas modernas de desenvolvimento *web*, utilizando HTTP e JSON como formato de troca de dados, permitindo independência tecnológica entre cliente e servidor.

14.3 Gestão do Estado do Jogo no *Frontend*

O JavaScript mantém variáveis de estado como *saldo, patamar, questionIndex*, *usedIds* e *currentQuestion*. Esta decisão evita chamadas desnecessárias ao servidor e permite uma experiência de utilizador mais fluida, mantendo no cliente apenas o estado visual e temporal, enquanto o estado lógico permanece no Prolog.

14.4 Temporizador por Pergunta

A implementação do temporizador (*setInterval*) introduz uma componente de pressão temporal, aproximando o jogo da versão real. Quando o tempo expira, é enviada uma resposta inválida ao Prolog, sendo corretamente interpretada como resposta errada. Esta decisão garante a reutilização da lógica existente no *backend*, evitando duplicação de regras.

14.5 Renderização Dinâmica da Interface

As perguntas e opções são criadas dinamicamente através do DOM (Document Object Model). Esta abordagem permite que o conteúdo seja totalmente controlado pelos dados recebidos do Prolog, garantindo consistência entre *backend* e *frontend* e evitando conteúdos estáticos rígidos.

14.6 Atualização de Saldo e Patamares

O *frontend* atualiza o saldo e o patamar após cada resposta correta, refletindo imediatamente o estado do jogo ao utilizador. A existência de patamares no *frontend* tem como objetivo apenas a apresentação visual, sendo coerente com as regras já implementadas no Prolog.

14.7 Implementação das Ajudas (Lifelines)

As ajudas (50/50, Ajuda do Público e Telefone) são implementadas exclusivamente em JavaScript. Esta decisão foi tomada por razões pedagógicas e de simplicidade, uma vez que estas funcionalidades não afetam a lógica central do jogo.

- 50/50: oculta duas opções incorretas
- Ajuda do Público: apresenta percentagens simuladas
- Telefone: sugere uma resposta com probabilidade controlada

A gestão local destas ajudas reduz a complexidade do *backend* e mantém a inferência lógica concentrada no Prolog.

14.8 *Feedback* Multimédia

O uso de sons e efeitos visuais no JavaScript visa melhorar a experiência do utilizador. A reprodução de áudio é controlada no *frontend* para evitar dependência excessiva do servidor e reduzir latência.

14.9 Controlo de Fluxo do Jogo

O JavaScript controla transições entre perguntas, bloqueio de opções após resposta, fim de jogo e reinício. Estas decisões garantem que o Prolog é apenas consultado quando necessário, reforçando a eficiência do sistema distribuído.

15. Justificação da Base de Conhecimento (`perguntas.pl`)

A base de conhecimento do sistema encontra-se definida no módulo perguntas, sendo responsável por armazenar todas as perguntas utilizadas no jogo. Esta separação é fundamental do ponto de vista académico e estrutural, pois isola o conhecimento declarativo da lógica procedimental.

15.1 Definição do Módulo

:- module(perguntas, [pergunta/6, todas_perguntas/1]).

A utilização de um módulo dedicado permite:

- Encapsular a base de conhecimento
- Controlar explicitamente os predicados exportados
- Facilitar manutenção, extensão e reutilização do conhecimento

São exportados apenas os predicados necessários ao funcionamento do sistema*: pergunta/6 e todas_perguntas/1.*

15.2 Estrutura do Predicado `pergunta/6`

Cada pergunta é representada por um facto com a seguinte estrutura:

pergunta(Id, Texto, Opcoes, RespostaCorreta, Nivel, Valor).

Esta modelação foi escolhida por ser clara, extensível e semanticamente adequada:

- *Id*: identificador único da pergunta
- *Texto*: enunciado da pergunta
- *Opcoes*: lista de pares letra–texto, facilitando a apresentação dinâmica
- *RespostaCorreta*: letra correspondente à opção correta
- *Nivel*: grau de dificuldade (1 a 5)
- *Valor*: prémio monetário associado

Esta representação segue o paradigma declarativo do Prolog, permitindo que o sistema raciocine sobre as perguntas sem necessidade de estruturas imperativas.

15.3 Organização por Níveis de Dificuldade

As perguntas estão organizadas por níveis crescentes de dificuldade, refletindo a progressão do jogo:

- Nível 1: perguntas simples e introdutórias
- Nível 2 e 3: conhecimento geral intermédio
- Nível 4: questões de maior complexidade
- Nível 5: pergunta final, de dificuldade elevada

Esta organização permite ao motor do jogo selecionar perguntas adequadas à fase atual, garantindo coerência pedagógica e progressão lógica.

15.4 Valores Associados às Perguntas

Cada pergunta possui um valor monetário crescente. Esta decisão aproxima o jogo da versão original e reforça a motivação do utilizador. Do ponto de vista técnico, a inclusão do valor diretamente no facto evita cálculos adicionais e permite que o *backend* devolva imediatamente a informação necessária ao *frontend*.

15.5 Predicado `todas_perguntas/1`

```
todas_perguntas(Lista) :-
    findall(pergunta(Id,Texto,Opcoes,Correta,Nivel,Valor),
        pergunta(Id,Texto,Opcoes,Correta,Nivel,Valor),
        Lista).
```

Este predicado permite obter a totalidade das perguntas definidas na base de conhecimento. A sua existência justifica-se por:

- Facilitar testes e validações
- Permitir futuras extensões (estatísticas, exportação, análise)
- Fornecer uma interface declarativa para acesso global ao conhecimento

15.6 Justificação Académica da Abordagem

A utilização de factos estáticos para representar conhecimento é uma das características fundamentais do Prolog. Esta base de conhecimento demonstra claramente:

- Separação entre conhecimento e inferência
- Clareza semântica
- Facilidade de extensão (novas perguntas podem ser adicionadas sem alterar o código do jogo)

Esta abordagem reforça o caráter académico do projeto, evidenciando boas práticas em programação lógica.

16. Criatividade

Identidade e Interface do Utilizador

A identidade visual do projeto foi consolidada sob a marca “MIAlionário”, uma simbiose criativa entre o título do jogo original e a sigla do mestrado (MIA). Para garantir a imersão e o dinamismo da experiência, o sistema inclui uma componente sonora com músicas e efeitos que acompanham as diversas fases da partida.

Ao nível da interface, o projeto cumpre os requisitos académicos através de uma abordagem dual:

- Conteúdo Temático: A base de conhecimento foi enriquecida com perguntas especificamente relacionadas com a unidade curricular de Lógica e Inteligência Artificial, reforçando a componente pedagógica do projeto.
- Modo Consola: Apresenta o logótipo do jogo em formato ASCII Art, assegurando o cumprimento das diretrizes de implementação.
- Versatilidade de Ambiente: O sistema permite que a partida decorra tanto em ambiente Web, através de um servidor HTTP e API REST, como diretamente na consola do SWI-Prolog.

17. Conclusão Final

O presente trabalho, desenvolvido no contexto do Mestrado em Inteligência Artificial, demonstrou de forma consistente e aprofundada a aplicação prática da programação lógica em Prolog, materializada no desenvolvimento de uma versão funcional e extensível do jogo “Quem Quer Ser Milionário”. Ao longo do projeto foi possível evidenciar que o Prolog, apesar de frequentemente associado a contextos académicos e teóricos, se revela plenamente capaz de sustentar aplicações modernas, interativas e integradas em arquiteturas distribuídas.

A solução apresentada assenta numa clara separação de responsabilidades entre a base de conhecimento, o motor de inferência lógica e as camadas de apresentação (consola e *web*), respeitando princípios fundamentais de engenharia de *software* e de sistemas baseados em conhecimento. A base de conhecimento declarativa, aliada a regras de inferência explícitas, permitiu modelar de forma transparente e rigorosa a lógica do jogo, incluindo progressão, patamares de segurança e validação de respostas.

A utilização de inferência lógica clássica, nomeadamente *Modus Ponens e Modus Tollens*, reforça o caráter académico do trabalho, permitindo demonstrar como mecanismos formais de raciocínio podem ser aplicados a problemas concretos. A inclusão intencional de uma falácia lógica contribui igualmente para uma reflexão crítica sobre os limites e potenciais erros em sistemas de inferência, enriquecendo o valor pedagógico do projeto.

A integração de um servidor HTTP e de uma API REST em Prolog evidencia a versatilidade da linguagem e a sua capacidade de interoperar com tecnologias *web* atuais. A comunicação em

formato JSON e a separação clara entre *frontend* (JavaScript) e *backend* (Prolog) seguem práticas modernas de desenvolvimento de *software*, garantindo modularidade, escalabilidade e facilidade de manutenção.

Adicionalmente, o projeto demonstra uma preocupação explícita com a experiência do utilizador, tanto no modo consola — através de *feedback* visual, cores ANSI e sons — como na interface *web*, com uma interação dinâmica, temporizada e visualmente apelativa. Estas escolhas mostram que é possível conjugar rigor lógico com aspetos práticos de usabilidade, sem comprometer a clareza do modelo declarativo subjacente.

Em síntese, este trabalho comprova que o Prolog é uma linguagem expressiva, poderosa e atual, capaz de suportar aplicações completas que combinam conhecimento declarativo, inferência lógica e integração com sistemas externos. Para além de cumprir os objetivos propostos, o projeto evidencia maturidade técnica, capacidade de abstração e uma abordagem criativa à programação lógica, constituindo um exemplo sólido da aplicação prática dos conceitos lecionados no âmbito do Mestrado em Inteligência Artificial.

Anexos

Poderão ser consultados em: https://github.com/MIA-CDFR/LIA.git