Aluno: Thauã Marcelo Souza Melelo
RA: 202313807
Disciplina: Inteligência Artificial

1. Introdução

Neste trabalho, eu implementei uma solução para o problema clássico do robô que deve sair de um labirinto. O robô começa na posição inicial (1,1) da matriz e precisa chegar até a posição final (10,10), desviando de obstáculos e cuidando da sua energia.

Decidi fazer esse trabalho como um jogo, utilizando a biblioteca Pygame, porque atualmente estou desenvolvendo um jogo por conta própria e gosto muito dessa área. Além disso, levar o problema para o visual me ajuda a raciocinar melhor, entender a lógica de forma mais clara e identificar possíveis erros ou melhorias.

Assim, além de cumprir os requisitos da disciplina, aproveitei para treinar minhas habilidades de programação visual e desenvolvimento de jogos.

O labirinto é gerado de forma **aleatória**, com obstáculos e pontos de energia espalhados no caminho. Para resolver o problema, utilizei o algoritmo de busca **BFS (Busca em Largura)**, que é eficiente para encontrar o caminho mais curto.

2. Como o problema foi resolvido

O problema pede algumas regras bem claras:

✅ O labirinto deve ter **obstáculos aleatórios**, entre **15 e 35** posições bloqueadas.
✅ O robô só pode se mover para **cima, baixo, esquerda e direita**.
✅ O robô perde **1 ponto de energia** a cada movimento, começando com **50 pontos**.
✅ Existem pontos de recuperação de energia:

- **5 posições** que recuperam **5 pontos**.
- **3 posições** que recuperam **10 pontos**.

O robô "morre" se a energia chegar a zero antes de concluir o percurso.

3. Como funciona o algoritmo

3.1. Representação do Labirinto

- O labirinto é uma **matriz de 10x10**, onde:
  - `0` representa posição livre.
  - `-1` representa um **obstáculo**.
- As posições (0,0) e (9,9) são o **início e fim**, e **nunca têm obstáculos**.
- Os **obstáculos** são sorteados aleatoriamente.

3.2. Pontos de energia

- As posições de energia também são **aleatórias**, mas só podem ser colocadas onde não há obstáculos nem na posição inicial ou final.
- Coloquei:
  - 5 posições de **+5 de energia**.
  - 3 posições de **+10 de energia**.

3.3. Algoritmo de busca

Usei o algoritmo de **Busca em Largura (BFS)**:

- Começa na posição inicial (0,0).
- Explora as posições vizinhas (cima, baixo, esquerda, direita).
- Guarda as posições visitadas para não voltar nelas.
- Quando chega na posição final (9,9), o caminho está completo.

Se o BFS **não encontrar um caminho** (porque o labirinto ficou impossível), o código gera outro labirinto, repetindo até encontrar um com caminho válido (no máximo 10 tentativas).

4. Controle da Energia

- O robô começa com **50 pontos**.
- Cada movimento custa **-1** de energia.
- Quando o robô passa por uma posição com **energia extra**, ele recupera:
    - **5 ou 10 pontos**, dependendo do tipo.
- O robô **não pode passar de 50 pontos** de energia máxima.
- Se a energia chegar a **0**, o robô "morre" e o caminho que ele percorreu até morrer é marcado em **vermelho**.

5. Como é exibido graficamente

O programa usa a biblioteca **Pygame** para mostrar o labirinto na tela. As cores são:

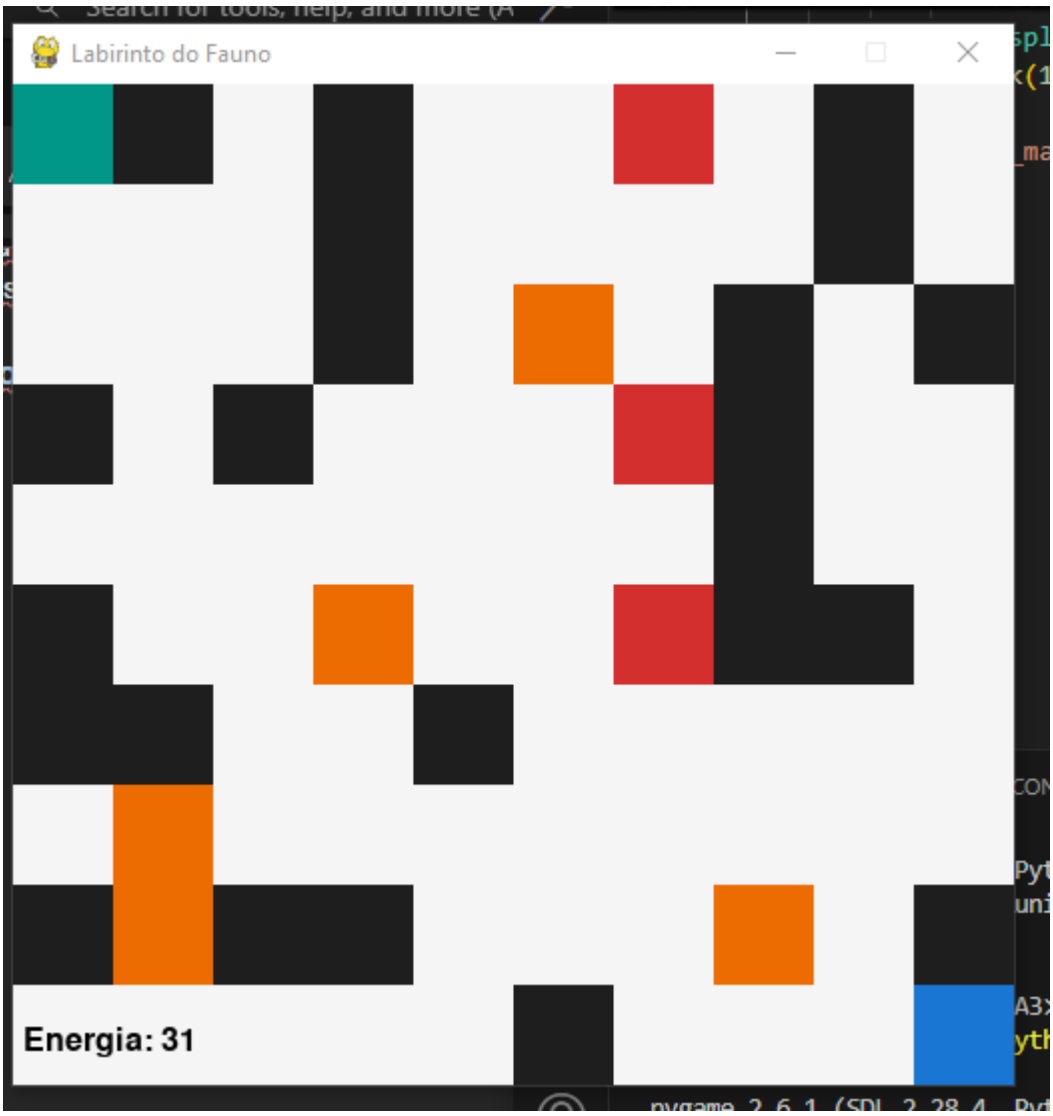
- **Caminho livre:** branco.
- **Obstáculos:** preto.
- **Início:** verde-água.
- **Fim:** rosa.
- **Robô:** azul.
- **Energia +5:** laranja.
- **Energia +10:** vermelho.
- **Caminho percorrido:** verde.
- **Caminho onde morreu:** vermelho.

O robô anda automaticamente pelo caminho encontrado.

6. Exemplos de labirintos gerados

Como o labirinto é **aleatório**, eu rodei o programa algumas vezes e peguei imagens dos resultados.

Exemplo 1: Robô chegou até o final



7. Código Fonte

O código está completo e bem comentado, explicando cada parte.
As funções principais são:

- **generate_maze():** cria o labirinto aleatório.
- **place_energy():** coloca os pontos de energia.
- **bfs():** encontra o caminho com busca em largura.
- **draw_grid():** desenha o labirinto e o robô na tela.
- **main():** executa tudo.

8. Conclusão

Esse trabalho me ajudou muito a entender como funciona a **IA simbólica**, e como implementar **algoritmos de busca** na prática.

Decidi transformar o problema em um jogo porque, além de estar desenvolvendo um jogo por conta própria, acredito que visualizar o problema graficamente me ajuda muito a raciocinar e perceber melhor o comportamento do algoritmo.

O algoritmo de **Busca em Largura** é ótimo para encontrar o caminho mais curto, mas ele não considera a energia do robô — ele só procura o caminho possível. Mesmo assim, com o controle de energia implementado, dá pra ver como decisões no caminho podem afetar a sobrevivência do robô.

Além disso, trabalhar com geração **aleatória** fez o labirinto ficar sempre diferente, tornando o problema mais desafiador e interessante.