

UNIVERSIDADE SÃO JUDAS TADEU



ANABELLY SALLES – 824129970
JHONATTAN MARIANO – 824120739
JOÃO GOBBI – 824145710
JULIA CRISTINA ARAUJO – 823157679
MATHEUS YUSUKE – 825146805
NICOLAS TRINDADE – 824135758

INTELIGÊNCIA ARTIFICIAL -
IA SIMBÓLICA

SÃO PAULO
2025

1.0 INTRODUÇÃO AO PROBLEMA

O presente trabalho tem como objetivo aplicar uma abordagem simbólica de Inteligência Artificial (IA) para resolver um problema de navegação em labirinto, onde um robô deve sair da posição inicial (1,1) até a posição final (10,10), enfrentando desafios como obstáculos, consumo de energia e regeneração parcial em alguns pontos. A proposta utiliza o algoritmo A* como principal método de busca do caminho ideal.

2.0 DESCRIÇÃO

2.1 O Robô:

- Começa na posição (1,1) com **50 unidades de energia**.
- **Perde 1 unidade** de energia por movimento.
- Pode recarregar energia passando por: **Energia +5** (representada por células verdes) e **Energia +10** (representada por células verde-escuras).
- O objetivo é alcançar a posição (9,9), evitando obstáculos e gerenciando sua energia.

2.2 Regras do Labirinto:

- **A matriz tem tamanho:** 10x10.
- **No labirinto existem:** O labirinto contém entre 15 a 35 obstáculos aleatórios, que são posições impassáveis. Existem 5 posições que restauram 5 pontos de energia cada e 3 posições especiais que restauram 10 pontos de energia.
- **O robô só pode mover-se:** cima, baixo, esquerda e direita.
- Se a energia zerar, o robô "morre".

3.0 ESTRUTURA DO CÓDIGO

3.1 Algoritmo Utilizado (A*)

Foi implementado o algoritmo A* (A-Star), que utiliza:

- Heurística da distância de Manhattan.
- Gerenciamento do consumo e recuperação de energia.
- Caminho gerado é sempre o mais eficiente possível (se existir um).

3.2 Geração do Labirinto

```
def generate_maze():
```

Cria uma matriz 10x10 contendo:

- Espaços livres (0).
- Obstáculos (-1)
- Energias +5 (5)
- Energias +10 (10)

Regras:

- Os obstáculos são posicionados aleatoriamente, variando entre 15 e 35.
- Energias +5: 5 unidades.
- Energias +10: 3 unidades.
- As posições de início (0, 0) e objetivo (9, 9) nunca são ocupadas.

Função Heurística (Distância de Manhattan)

```
def heuristic(a, b):
```

Calcula a distância de Manhattan entre dois pontos (a, b), utilizada pelo A* para estimar o custo restante até o objetivo.

3.3 Algoritmo A Adaptado com Energia

```
def astar(maze):
```

Modificação do algoritmo A* tradicional, incorporando:

- Consumo de energia a cada movimento (-1).
- Reabastecimento ao passar sobre células de energia (+5 ou +10).

A fronteira (frontier) armazena tuplas com:

- Função de custo total $f = g + h$.
- Custo real até o ponto g.
- Energia restante.
- Posição atual.
- Caminho percorrido até o momento.








Condições de parada:

- Se atingir o objetivo (9,9), retorna o caminho e a energia restante.
- Se a energia zerar antes de chegar, retorna None.

3.4 Visualização do Labirinto

```
def plot_maze(maze, path):
```

Gera uma imagem colorida do labirinto com:

-  Objetivo (9, 9) em vermelho.
-  Início (0, 0) em amarelo.
-  Caminho percorrido em azul.
-  Obstáculos em preto.
-  Energia +5 em verde claro.
-  Energia +10 em verde escuro.
-  Espaços livres em branco.

Essa etapa cria uma função para mostrar a visualização via matplotlib.

3.5 Execução Principal

```
maze = generate_maze()

path, energy_left = astar(maze)

if path:

    plot_maze(maze, path)

    print(f"Caminho percorrido: {path}")

    print(f"Passos: {len(path)}, Energia restante: {energy_left}")

else:

    plot_maze(maze, []) # opcional: ainda pode mostrar o labirinto vazio

    print("Labirinto gerado é impossível de ser resolvido")
```

Processo:

- Gera um labirinto aleatório.
- Executa o A* considerando energia.
- Plota o labirinto e o caminho encontrado (ou nenhum caminho, se falhar).

Imprime no console:

Labirinto gerado é solucionado:

- O caminho percorrido.
- O número de passos.
- A energia restante ao final.

Labirinto gerado é impossível de ser solucionado

- Quando os obstáculos gerados, criam uma barreira impossível do robô passar, a seguinte mensagem é apresentada: "Labirinto gerado é impossível de ser resolvido".

4.0 CÓDIGO EM PYTHON

Repositorio GitHub: https://github.com/Japu431/A3-Inteligencia_Artificial

```
import random
import heapq
import matplotlib.pyplot as plt
import numpy as np

# Constantes
N = 10
ENERGY_START = 50
ENERGY_5_COUNT = 5
ENERGY_10_COUNT = 3
OBSTACLE_MIN = 15
OBSTACLE_MAX = 35

# Representações
CLEAR = 0
OBSTACLE = -1
ENERGY_5 = 5
ENERGY_10 = 10
START = (0, 0)
GOAL = (9, 9)

# Geração do labirinto
def generate_maze():
    maze = [[CLEAR for _ in range(N)] for _ in range(N)]

    # Obstáculos
    obstacles = set()
    while len(obstacles) < random.randint(OBSTACLE_MIN, OBSTACLE_MAX):
        x, y = random.randint(0, N-1), random.randint(0, N-1)
        if (x, y) not in [START, GOAL]:
            maze[x][y] = OBSTACLE
            obstacles.add((x, y))

    # Energia +5
    energy5 = set()
    while len(energy5) < ENERGY_5_COUNT:
        x, y = random.randint(0, N-1), random.randint(0, N-1)
        if maze[x][y] == CLEAR and (x, y) not in [START, GOAL]:
            maze[x][y] = ENERGY_5
            energy5.add((x, y))

    # Energia +10
    energy10 = set()
```

```

while len(energy10) < ENERGY_10_COUNT:
    x, y = random.randint(0, N-1), random.randint(0, N-1)
    if maze[x][y] == CLEAR and (x, y) not in [START, GOAL] and (x, y) not in
energy5:
        maze[x][y] = ENERGY_10
        energy10.add((x, y))

    return maze

# Heurística de Manhattan
def heuristic(a, b):
    return abs(a[0] - b[0]) + abs(a[1] - b[1])

# Algoritmo A* com energia
def astar(maze):
    start = START
    goal = GOAL
    frontier = [(0, 0, ENERGY_START, start, [])]
    visited = set()

    while frontier:
        f, g, energy, current, path = heapq.heappop(frontier)
        if current in visited:
            continue
        visited.add(current)
        path = path + [current]

        if current == goal:
            return path, energy

        if energy <= 0:
            continue

        for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:
            x2, y2 = current[0] + dx, current[1] + dy
            if 0 <= x2 < N and 0 <= y2 < N and maze[x2][y2] != OBSTACLE:
                pos = (x2, y2)
                if pos in visited:
                    continue
                next_energy = energy - 1
                if maze[x2][y2] == ENERGY_5:
                    next_energy += 5
                elif maze[x2][y2] == ENERGY_10:
                    next_energy += 10
                h = heuristic(pos, goal)
                heapq.heappush(frontier, (g + 1 + h, g + 1, next_energy, pos, path))

```

```

    return None, 0

# Visualizar o labirinto
def plot_maze(maze, path):
    img = np.zeros((N, N, 3), dtype=np.uint8)

    for i in range(N):
        for j in range(N):
            if maze[i][j] == OBSTACLE:
                img[i, j] = [0, 0, 0]
            elif maze[i][j] == ENERGY_5:
                img[i, j] = [0, 255, 0]
            elif maze[i][j] == ENERGY_10:
                img[i, j] = [0, 128, 0]
            else:
                img[i, j] = [255, 255, 255]

    for (x, y) in path:
        img[x, y] = [0, 0, 255]

    img[START[0], START[1]] = [255, 255, 0]
    img[GOAL[0], GOAL[1]] = [255, 0, 0]

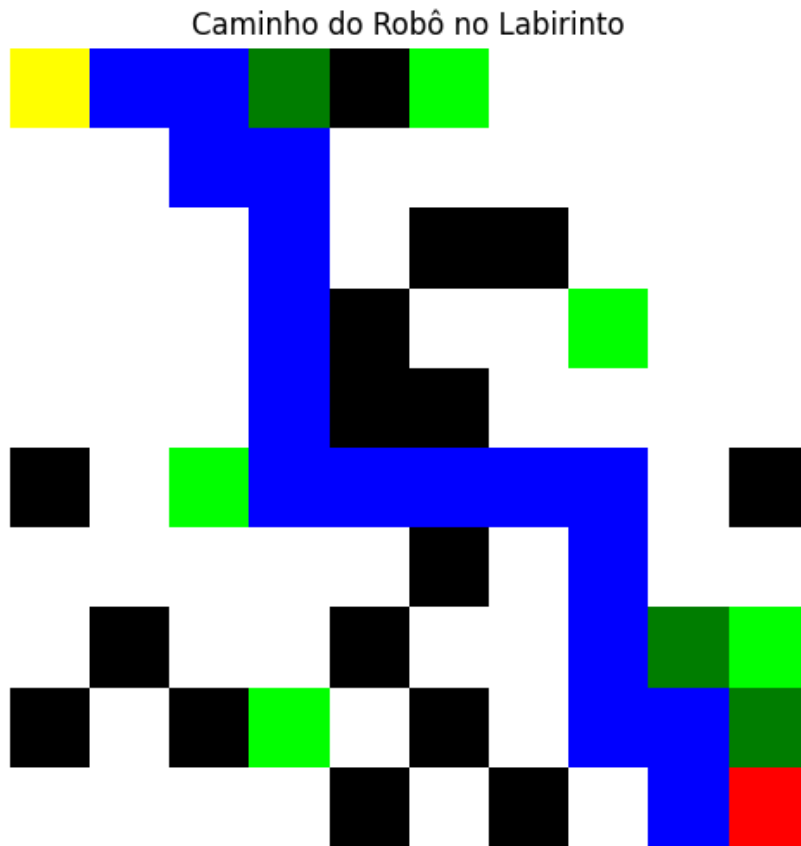
    plt.figure(figsize=(6, 6))
    plt.imshow(img)
    plt.title("Caminho do Robô no Labirinto")
    plt.axis('off')
    plt.show()

# Executar
maze = generate_maze()
path, energy_left = astar(maze)
if path:
    plot_maze(maze, path)
    print(f"Caminho percorrido: {path}")
    print(f"Passos: {len(path)}, Energia restante: {energy_left}")
else:
    plot_maze(maze, [])
    print("Labirinto gerado é impossível de ser resolvido")

```


5.0 EXEMPLOS DE EXECUÇÃO

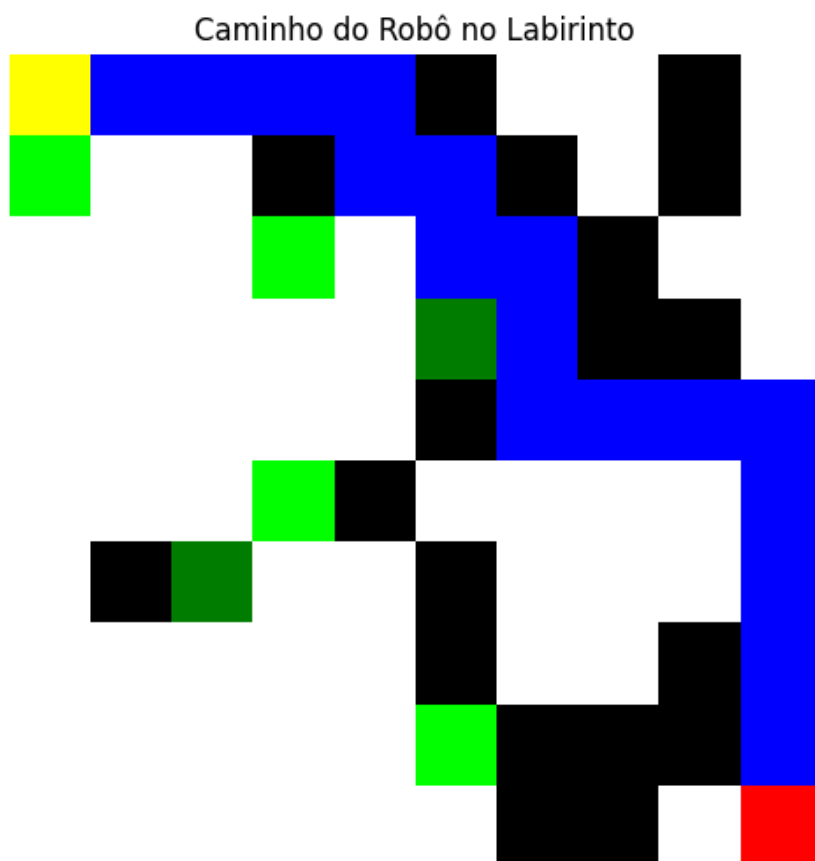
Exemplo 1:



Passos: 19, Energia restante: 37

Caminho percorrido: [(0, 0), (1, 0), (1, 1), (1, 2), (1, 3), (2, 3), (2, 4), (2, 5), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (6, 7), (6, 8), (6, 9), (7, 9), (8, 9), (9, 9)]

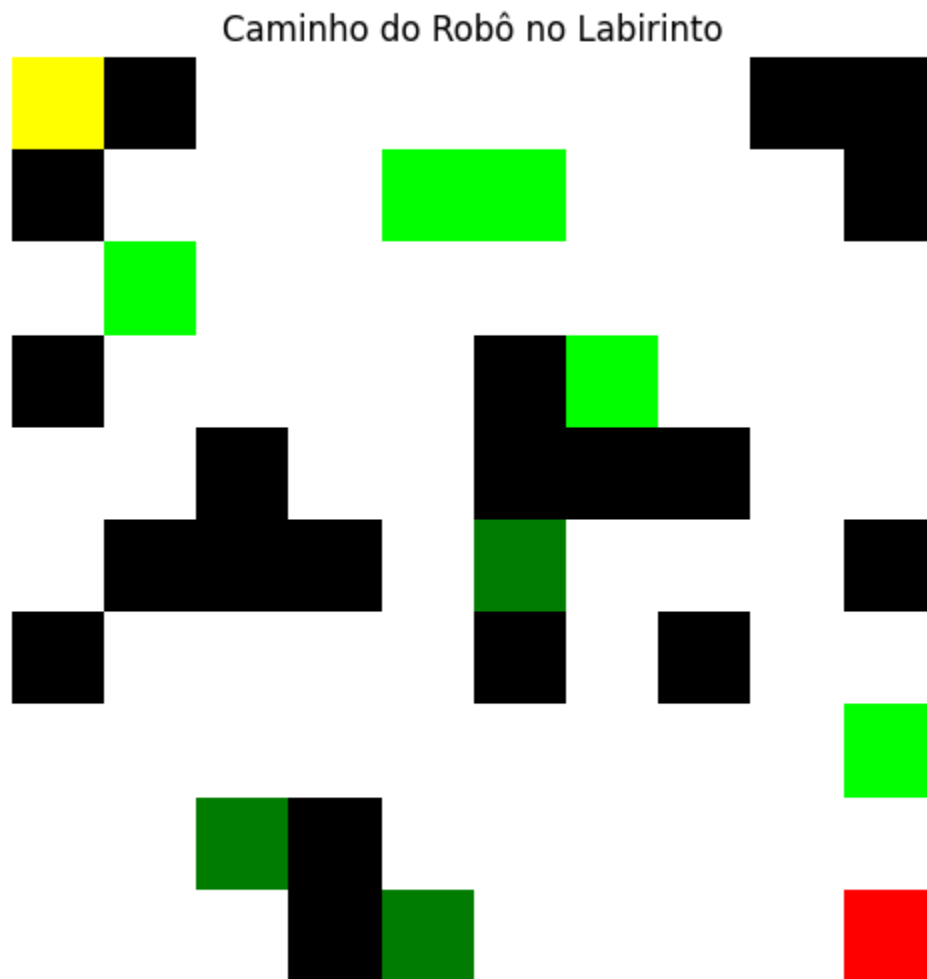
Exemplo 2:



Passos: 19, Energia restante: 47

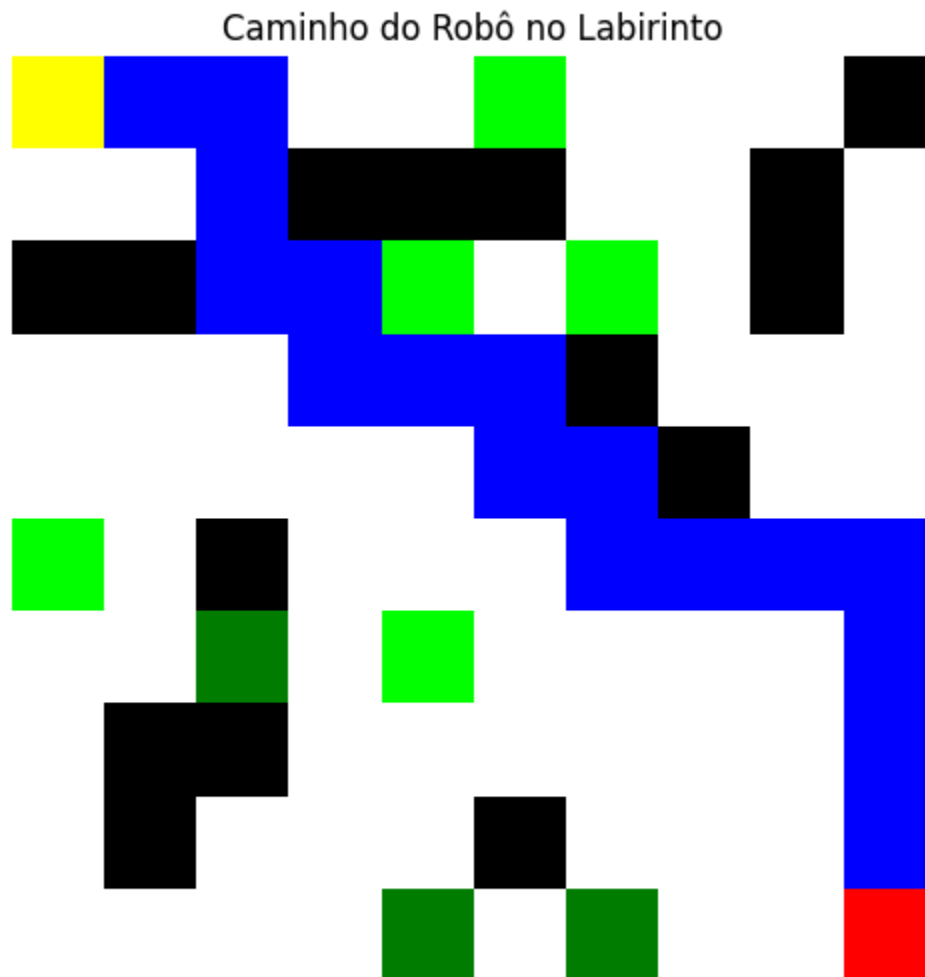
Caminho percorrido: [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (1, 5), (2, 5), (2, 6), (3, 6), (4, 6), (4, 7), (4, 8), (4, 9), (5, 9), (6, 9), (7, 9), (8, 9), (9, 9)]

Exemplo 3



Labirinto gerado é impossível de ser resolvido

Exemplo 4:



Passos: 19, Energia restante: 32

Caminho percorrido: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3), (3, 3), (3, 4), (3, 5), (4, 5), (4, 6), (5, 6), (5, 7), (5, 8), (5, 9), (6, 9), (7, 9), (8, 9), (9, 9)]