Solucionador de Labirintos

1. Visão Geral do Projeto

Este projeto consiste em uma aplicação gráfica interativa que permite ao usuário criar um labirinto e visualizar a solução através de algoritmos de busca. O objetivo principal é demonstrar de forma visual como os algoritmos de Busca em Largura (BFS) e Busca em Profundidade (DFS) exploram um espaço de estados para encontrar um caminho entre um ponto inicial e um final.

2. Como Foi Desenvolvido

A aplicação foi desenvolvida utilizando as seguintes tecnologias:

- **Linguagem de Programação:** Python
- **Biblioteca Gráfica:** Pygame, uma biblioteca de código aberto para a criação de aplicações multimídia e jogos em Python. Ela foi utilizada para:
    - Desenhar a grade do labirinto na tela.
    - Renderizar as células com cores diferentes para representar caminhos, paredes, pontos de início/fim, células visitadas e o caminho final.
    - Capturar e processar as interações do usuário, como cliques do mouse para desenhar paredes e comandos do teclado para iniciar a busca.

O labirinto em si é representado por uma estrutura de dados de matriz em Python, onde cada elemento corresponde a uma célula na grade.

3. Algoritmos Utilizados

O solucionador implementa dois algoritmos clássicos de **busca não informada (busca cega)**:

1. **Busca em Largura (BFS - *Breadth-First Search*):** Este algoritmo explora todos os vizinhos de um nó antes de continuar para os vizinhos dos vizinhos. Ele avança de forma uniforme a partir do ponto inicial, garantindo que o caminho encontrado seja o mais curto em número de passos.
2. **Busca em Profundidade (DFS - *Depth-First Search*):** Este algoritmo explora um ramo do labirinto o mais fundo possível até atingir um beco sem saída, para então retroceder (*backtracking*) e tentar outro caminho. Ele é eficiente em termos de memória, mas não garante a descoberta do caminho mais curto.

Dentre as várias funções presentes no arquivo, as funções ***start_search()*** e ***step()*** são as que de fato executam a DFS ou BFS. Para facilitar a implementação híbrida,

foi utilizado a estrutura de dados ***deque*** do Python, que permite implementar tanto filas quanto pilhas de forma eficiente.

O usuário pode executar qualquer um dos algoritmos e observar a animação do processo de busca, o que ajuda a entender as diferenças fundamentais na estratégia de exploração de cada um. É possível alterar o tamanho do labirinto, a velocidade da animação, entre outros parâmetros.

```python
# Configurações visuais
# Edite CELL_SIZE e GRID_ROWS/COLS para ajustar o tamanho do labirinto
# Tamanhos padrão: 30x30 células de 20px
CELL_SIZE = 20  # diminua o tamanho para labirintos maiores
MARGIN = 2
GRID_ROWS = 30
GRID_COLS = 30
FOOTER_HEIGHT = 70  # espaço para texto
FOOTER_WIDTH = 10
FPS = 60  # taxa de frames
# passos de busca por frame (aumente para acelerar em grades grandes como 500x100)
ANIMATION_SPEED = 5      You, há 12 minutos • busca não informada
DENSITY = 0.3  # densidade de paredes ao preencher aleatoriamente o labirinto com paredes

# Cores RGB
WHITE = (255, 255, 255)
BLACK = (0, 0, 0)
GRAY = (200, 200, 200)
GREEN = (0, 200, 0)       # início
RED = (200, 0, 0)         # fim
YELLOW = (255, 255, 0)    # visitadas
BLUE = (50, 50, 255)      # caminho final
LIGHT_BLUE = (150, 200, 255)  # nó atualmente explorado (frontier)
```

A imagem abaixo exibe a interface do projeto. Uma grade colorida representa o labirinto, onde em preto temos as paredes, em branco células não visitadas, em amarelo células visitadas, em azul células conhecidas, porém não visitadas. Por último, em roxo há o caminho encontrado pelo algoritmo.