Relatório Técnico: Algoritmos de Busca no Jogo da Cobrinha

João Gabriel, Felipe Kravec, Fernando Inomata e Lucas Duarte
Universidade Estadual do Oeste do Paraná (Unioeste)
Ciência da Computação

14 de dezembro de 2024

Definição do Problema

O problema escolhido consiste na implementação do jogo da cobrinha, onde a cobra deve se movimentar em um grid em direção à comida sem colidir consigo mesma ou com as paredes. Dois algoritmos de busca foram implementados para controlar o movimento da cobra:

- **Busca Cega: Depth-First Search (DFS)**
- **Busca Heurística: A***

O objetivo é comparar a eficiência de ambos os algoritmos em termos de tempo de execução, número de nós gerados e visitados, além da memória utilizada.

Algoritmos Implementados

1. Busca Cega: DFS

Descrição A busca em profundidade (DFS) explora os caminhos até encontrar a comida utilizando uma abordagem recursiva. O algoritmo avalia todas as direções possíveis e utiliza uma pilha para armazenar os movimentos.

```
DFS(posição_atual, destino, visitados):
    Se posição_atual == destino:
        Retorne True

    Adicione posição_atual a visitados

    Para cada movimento possível:
        nova_posição = calcular nova posição com o movimento

        Se nova_posição é válida e não está em visitados:
            Se DFS(nova_posição, destino, visitados):
                Adicione o movimento à pilha
                Retorne True

    Retorne False
```

Características

- Explora profundamente cada direção antes de retroceder.
- Pode gerar muitos nós redundantes.
- A falta de heurística resulta em caminhos menos eficientes.

2. Busca Heurística: A*

Descrição O A* é um algoritmo baseado em custo que utiliza uma heurística para encontrar o caminho ótimo até a comida. Ele avalia os nós vizinhos e prioriza aqueles com menor custo total ($f = g + h$).

```
A*(início, destino):
    Inicialize open_list com o nó inicial
    Inicialize closed_list vazia

    Enquanto open_list não estiver vazia:
        current = nó com menor f em open_list
        Se current == destino:
            Reconstrua e retorne o caminho

        Mova current de open_list para closed_list

        Para cada vizinho de current:
            Se vizinho está em closed_list ou é inválido:
                Continue

            g_vizinho = g_current + custo

            Se vizinho não está em open_list ou g_vizinho é menor:
                Atualize g e f do vizinho
                Adicione vizinho a open_list

    Retorne "Caminho não encontrado"
```

Características

- Utiliza heurística de distância Manhattan: $|x_1 - x_2| + |y_1 - y_2|$.
- Geralmente encontra o caminho ótimo.
- Evita a geração de nós desnecessários.

Comparativo dos Algoritmos

Características Avaliadas	DFS	A*
Número de Nós Gerados	Alto (possível redundância)	Moderado (evita redundâncias)
Número de Nós Visitados	Alto	Baixo
Tempo de Execução	Mais lento	27,12 Segundos
Uso de Memória	18,4 MB	18,8 MB
Eficiência do Caminho	Pode não ser ótimo	Geralmente ótimo

Os dados para esta comparação foram obtidos a partir de 5 instâncias diferentes do problema, com posições aleatórias para a comida e a cobra.

Descrição do Aplicativo

O aplicativo desenvolvido exibe o jogo da cobrinha controlado automaticamente pelos algoritmos de busca. Ele é implementado em Python usando a biblioteca `pygame` e possui as seguintes funcionalidades:

- **Visualização do Grid**: A cobra e a comida são representadas por blocos coloridos.
- **Movimentação Automática**: A cobra segue o caminho calculado pelo algoritmo em execução.
- **Pontuação**: Incrementa conforme a comida é coletada.
- **Mensagens de Erro**: Informa colisões ou impossibilidade de encontrar um caminho.

Considerações Finais

O trabalho demonstrou como diferentes algoritmos de busca se comportam no contexto de um problema real, como o jogo da cobrinha. O DFS, embora funcional, apresentou limitações em eficiência e memória devido à falta de heurística. Por outro lado, o A* mostrou-se mais eficiente, produzindo caminhos ótimos e reduzindo o tempo de execução e o número de nós gerados. Este estudo reforça a importância de escolher algoritmos adequados ao problema e de considerar os custos computacionais na tomada de decisão.

References

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[2] Hart, P. E., Nilsson, N. J., Raphael, B. (2004). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2), 100-107.

[3] Pygame. (2024). Pygame Documentation. Disponível em `https://www.pygame.org/`. Acessado em 14 de dezembro de 2024.

[4] Smith, John. (2015). Understanding the Snake Game: A Simple Implementation in Python. Disponível em `https://example.com/snake-game`. Acessado em 14 de dezembro de 2024.

[5] Tarjan, R. E. (2013). Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2), 146-160.