



## **Relatório**

### **Trabalha Prático de Algoritmo e Estrutura de Dados 2**

Problema 3: Dogo

Aluno: Alexandre Gabriel Gadelha de Lima



## **Índice**

1. Descrição do problema **3**
2. Explicação do procedimento de busca implementado **5**
3. Discussão sobre as decisões de projeto para a elaboração de interface **6**
4. As instruções de instalação e uso do aplicativo desenvolvido **8**
5. Referências.**10**



**Poder Executivo**  
**Ministério da Educação**  
**Universidade Federal do Amazonas**  
**Instituto de Computação**

---





**Poder Executivo**  
**Ministério da Educação**  
**Universidade Federal do Amazonas**  
**Instituto de Computação**

---



**UFAM**



## Explicação do procedimento de busca implementado

O algoritmo utilizado é o A\* (A estrela), um algoritmo de busca heurística que encontra o caminho mais curto de um ponto de início para um ponto de objetivo em um grafo ou grid.

- **Inicialização:**
  - O algoritmo começa inicializando uma fila de prioridade chamada *agenda* que será usada para manter os estados a serem explorados. O estado inicial é adicionado à *agenda* com um custo de 0.
- **Exploração de Estados:**
  - Enquanto a *agenda* não estiver vazia, o algoritmo continua explorando estados.
  - Em cada iteração, o algoritmo remove o estado com o menor custo acumulado da *agenda*. Este estado é o candidato atual para ser explorado.
- **Verificação do Objetivo:**
  - O meu algoritmo verifica se o estado atual é o estado de objetivo. Se for, o caminho é reconstruído a partir do dicionário *caminho* (que rastreia os estados anteriores) e é retornado como o caminho mais curto encontrado.
- **Geração de Estados Adjacentes:**
  - Se o estado atual não for o estado de objetivo, o algoritmo gera estados adjacentes ao estado atual. Os estados adjacentes são gerados movendo-se nas direções cima, baixo, esquerda e direita a partir do estado atual.
  - Para cada estado adjacente gerado, o algoritmo verifica se o estado é válido (dentro dos limites do grid e não um obstáculo). Se o estado é válido e ainda não foi explorado, ele é adicionado à *agenda* com um custo total calculado usando a heurística A\*.
- **Cálculo do Custo:**
  - O custo total de um estado é calculado usando a função *salao.custo\_total(estado, objetivo)*. Esta função inclui um custo base, que é a distância em grade do estado ao objetivo. Além disso, há um custo adicional para estados adjacentes a obstáculos, incentivando o algoritmo a evitar esses estados.
- **Exploração e Rastreamento:**
  - O algoritmo continua explorando estados adjacentes válidos até encontrar o estado de objetivo. Durante esse processo, ele rastreia os estados anteriores no dicionário *caminho* para reconstruir o caminho mais curto quando o estado de objetivo é alcançado.
- **Retorno do Caminho:**
  - Quando o estado de objetivo é alcançado, o algoritmo reconstrói o caminho do estado de objetivo ao estado inicial usando o dicionário *caminho*. O caminho é então invertido para obter a ordem correta do início ao objetivo.



Em resumo, o algoritmo A\* funciona explorando estados candidatos de acordo com uma função de custo que leva em consideração tanto o custo acumulado do início ao estado atual quanto uma heurística que estima o custo do estado atual ao objetivo. Ele continua explorando estados até encontrar o estado de objetivo ou até que não haja mais estados na **agenda** para explorar.

### **Discussão sobre as decisões de projeto para a elaboração de interface**

A biblioteca utilizada para fazer a interface gráfica do projeto foi a **pygame**, optei por ela por ser uma biblioteca com boa quantidade de tutoriais e materiais disponíveis e achei de simples uso, atendeu todas as necessidades do projeto sem.

- **Uso de janelas gráficas**
  - O código gera janelas gráficas para criar uma interface interativa com o usuário.
  - É por meio de janelas que o programa disponibiliza mensagens, instruções e entradas do usuário, seja para escolher as dimensões ou posição de obstáculos.
- **Modos de Jogo:**
  - O programa permite ao usuário escolher entre os modos de jogo "manual" e "automático". Isso é feito por meio de uma interface gráfica que exibe opções para o usuário.
- **Escolha de Cores, ícones e fonte:**
  - O código usa uma paleta de cores definida (PRETO, BRANCO, VERMELHO, AZUL, CINZA\_ESCURO) para garantir uma aparência visual agradável e contrastante. Cores diferentes são usadas para representar diferentes elementos na interface, como o salão, obstáculos, mensagens, etc.
  - Me inspirei na paleta de cores do tema AYU do Visual Studio Code
  - Como é bastante evidente o caráter mais geométrico da interface optei por uma fonte no padrão 8 bits.
- **Botões e Clique do Mouse:**
  - O código permite que o usuário faça escolhas clicando em opções na tela. Por exemplo, ao escolher entre "Modo Manual" e "Modo Automático", o usuário pode clicar diretamente no botão correspondente na tela. A posição do mouse é detectada para determinar qual opção o usuário selecionou.



- Optei também por fazer a escolha da posição de obstáculos através de uma simulação do salão que reage ao clique de usuário gerando um feedback visual
- **Tutorial Interativo:**
  - Na versão inicial do programa, submeti-o a testers (minha irmã e mãe), a fim de coletar feedback e aprimorar a experiência do usuário. Durante esse processo, identifiquei que algumas entradas incorretas por parte dos usuários poderiam ser resolvidas por meio de tutoriais explicativos integrados às opções disponíveis no programa. Um dos principais pontos de atenção foi observado na tela de seleção da posição dos obstáculos, onde os usuários demonstraram dificuldades em entender completamente o procedimento.
  - O programa apresenta um tutorial interativo explicando como o usuário pode definir os obstáculos manualmente. O tutorial é exibido em texto na tela e pode ser pulado pressionando a barra de espaço.
- **Obstáculos e adjacentes**
  - Inicialmente, a abordagem escolhida envolvia a geração gráfica dos obstáculos juntamente com suas células adjacentes. No entanto, deparando-me com desafios significativos nesse processo, decidi modificar a estratégia. Optei por primeiro gerar os obstáculos de forma isolada e, posteriormente, implementar uma função responsável por identificar esses obstáculos e destacar suas células adjacentes com uma cor distinta.
- **Mensagens de Status:**
  - O programa exibe mensagens de status na tela em resposta às ações do usuário. Por exemplo, mensagens como "Game Over" e "Good Job!" são exibidas quando o usuário atinge determinadas condições de vitória ou derrota.



## As instruções de instalação e uso do aplicativo desenvolvido

### Instalação:

Para executar este programa, você precisará das seguintes bibliotecas e recursos instalados no seu ambiente Python:

1. **Python:** Primeiro, você precisa ter o Python instalado no seu sistema. Você pode baixar a versão mais recente do Python em [python.org](https://python.org).
2. **Pygame:** O código usa a biblioteca Pygame para criar a interface gráfica. Você pode instalar o Pygame usando o *pip*, que é o gerenciador de pacotes do Python. Execute o seguinte comando no seu terminal ou prompt de comando para instalar o Pygame:

```
pip install pygame
```

3. Dependendo do sistema operacional, você pode executar o script Python no terminal (Linux/macOS) ou no prompt de comando (Windows) usando o seguinte comando, assumindo que o já encontrou o diretório do programa *main.py*:

```
python main.py
```

ou

```
py main.py
```

4. Isso iniciará a execução do programa e abrirá uma janela com o jogo "Ninas's Adventure"

Outra opção inclui dar *run* no arquivo *main.py* utilizando uma IDE com suporte a python, como por exemplo o PyCharm.

### Uso:

1. Após executar o programa, uma janela irá perguntar se o usuário deseja escolher as dimensões ou se prefere da forma que o programa está configurado
  - 1.1. Entrada esperada: Teclas **S** ou **N** serem pressionadas
2. Se o usuário escolher por definir as dimensões do programa, uma nova tela perguntando as dimensões irá aparecer.
  - 2.1. Entrada esperada: **Um número inteiro** ser digitado e em seguida a tecla **Enter** ser pressionada
3. Será apresentada uma tela de escolha de obstáculos, onde o usuário poderá escolher se ele quer definir as posições dos obstáculos ou se quer que o programa gere aleatoriamente os obstáculos.





- 3.1. Entrada esperada: **Clique** com mouse nas opções “**Escolher obstáculos**” ou em “**Gerar obstáculos**”.
4. Se o usuário clicar em “Escolher obstáculos”, uma tela de tutorial irá aparecer, para fechar o tutorial o usuário deve pressionar a tecla **Espaço**.
  - 4.1. Após o tutorial, para escolher a posição o usuário deve **clicar** na posição referente a onde ela quer que o obstáculo se posicione, para desfazer a seleção ele deve **clicar novamente** que o obstáculo será desmarcado, para salvar as posições escolhidas o usuário deve clicar em **Enter**.
5. O usuário deverá escolher entre os modos Automático e Manual, no modo automático o algoritmo criará o caminho do dogo até o osso, no modo manual o usuário deverá mover o cachorro até o osso.
  - 5.1. Entrada esperada: **Clique** com mouse nas opções “**Modo Manual**” ou em “**Modo Automático**”.
6. Se no modo manual foi o escolhido então o usuário deve guiar o dogo até o osso utilizando as **setas do teclado**, se entrar em contato com as zonas dos obstáculos o usuário perde e recebe uma mensagem de “game over”, se chegar com sucesso até o osso receberá uma mensagem de “good job”



## Referências

O python file astar tem trechos de código baseados neste endereço

<https://stackoverflow.com/questions/72259253/optimization-of-a-implementation-in-python>

Principalmente as funções `reconstruir_caminho` e `a_star`

Trechos da lógica do algoritmo a\* para buscar foram usados no código

<https://stackoverflow.com/questions/72735205/convertng-dijkstra-to-a-star-algorithm-in-python>

Usei como base a estrutura do método fila de prioridade e também a forma como ele implementa no uso

<https://acervolima.com/fila-prioritaria-em-python/>

<https://ichi.pro/pt/como-implementar-facilmente-a-fila-prioritaria-em-python-227236962803094>