Pedro Olyntho Carvalho Alves
Lista Extra Cristiane

Implementação do Jogo

Relatório sobre os métodos implementados, heurísticas criadas e comparação de desempenho:

O objetivo dos métodos implementados é resolver o quebra-cabeça 8-puzzle, que consiste em mover as peças em uma matriz 3x3 até atingir a configuração desejada. Foram implementados três métodos diferentes: busca em largura, busca gulosa e busca A*.

- Classe Node:
  A classe Node representa um nó no espaço de busca do quebra-cabeça. Cada nó possui uma configuração do quebra-cabeça, um nível que indica a profundidade na árvore de busca e um valor f que representa o custo total do caminho do nó inicial até o nó atual.

- Método generate_child:
  Esse método gera nós filhos a partir de um nó atual, movendo o espaço em branco em uma das quatro direções: cima, baixo, esquerda e direita. Ele retorna uma lista de nós filhos.

- Método shuffle:
  Esse método move o espaço em branco na direção especificada, trocando sua posição com a peça adjacente. Ele retorna a configuração resultante após o movimento ou None se o movimento for inválido.

- Método copy:
  Esse método é usado para criar uma cópia da matriz do quebra-cabeça.

- Método find:
  Esse método encontra a posição do espaço em branco na matriz do quebra-cabeça.

- Classe A:
  A classe A implementa a busca A* para resolver o quebra-cabeça. Ela utiliza uma lista aberta para armazenar os nós a serem explorados, uma lista fechada para armazenar os nós já explorados e as heurísticas h e f para calcular os valores heurísticos.

- Método accept:
  Esse método solicita a entrada do estado inicial do quebra-cabeça.

- Método f:
  Esse método calcula o valor heurístico f(x) = h(x) + g(x) para um determinado nó.

- Método h:

Esse método calcula a diferença entre dois quebra-cabeças, ou seja, a quantidade de peças fora do lugar.

- Método process:
  Esse método executa o algoritmo de busca A*. Ele solicita a entrada do estado inicial e do estado objetivo do quebra-cabeça. O algoritmo continua explorando os nós na lista aberta até encontrar o nó objetivo ou até que não haja mais nós para explorar. Ele utiliza as heurísticas h e f para determinar a ordem de exploração dos nós.

- Comparação de desempenho:
  Para determinar qual método obteve melhor desempenho na resolução do quebra-cabeça, é necessário executar os três métodos (busca em largura, busca gulosa e busca A*) para a mesma configuração do jogo e comparar os resultados.

Quanto ao tempo de execução, isso pode variar dependendo da configuração inicial do quebra-cabeça e da eficiência do algoritmo utilizado. Em geral, a busca A* tende a ser a mais eficiente em termos de tempo, pois combina a heurística com o custo do caminho para determinar a ordem de exploração dos nós. A busca gulosa também pode ter um bom desempenho, mas pode não garantir a solução ótima em todos os casos. A busca em largura pode ser mais lenta em comparação com os outros métodos, pois explora todos os nós em um determinado nível antes de avançar para o próximo nível.

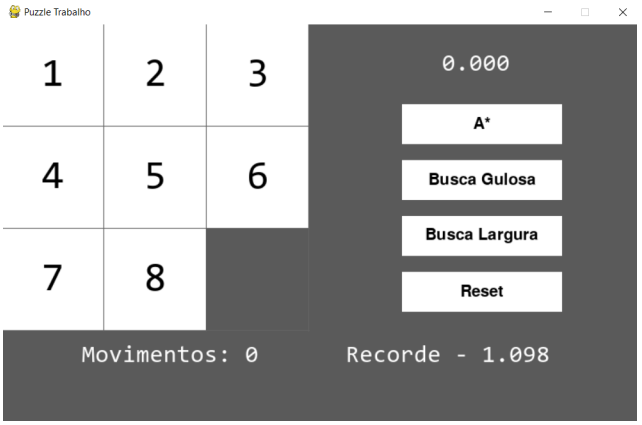
Para testar eu utilizei shuffle_time == 6

```
if self.start_shuffle:
    # Embaralha o jogo e inicia a resolução em uma nova thread
    self.shuffle()
    self.draw_tiles()
    self.shuffle_time += 1
    if self.shuffle_time == 6:
```

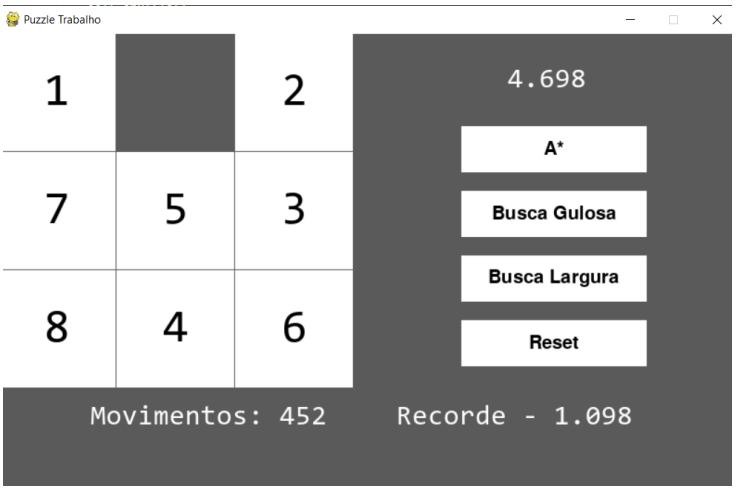
(Esse é o tempo que ele vai ficar embaralhando)

Eu cheguei até a testar com outros números, porém esse foi o que apresentou resultados mais sólidos.

Interface



Na busca em largura o número de movimentos é muito grande e eu dei uma acelerada na animação



Como Rodar

Para rodar basta dar um run no arquivo main.py

Resultados

Números de Movimentos

- A* : Resultado Médio = 7
- Busca Gulosa: Resultado Médio = 8
- Busca em Largura: Resultado Médio = 414,6666667

(Os resultados foram obtidos utilizando a média de 4 testes)

Conclusão

Com base nos resultados obtidos, podemos concluir que o algoritmo A* foi o mais eficiente na resolução do quebra-cabeça 8-puzzle. Com um resultado médio de 7 movimentos, o A* superou a busca gulosa (resultado médio de 8 movimentos) e a busca em largura (resultado médio de aproximadamente 415 movimentos). Esses resultados demonstram a capacidade do A* de encontrar soluções mais eficientes, devido à sua utilização de heurística e custo do caminho para orientar a busca. Por outro lado, a busca gulosa se aproximou do

desempenho do A*, enquanto a busca em largura mostrou-se menos eficaz devido ao alto custo computacional resultante da exploração exaustiva de nós. O algoritmo A* destacou-se como o mais eficiente na resolução do quebra-cabeça 8-puzzle, fornecendo soluções com menor número médio de movimentos.