|  | **Universidade do Oeste Paulista**  **Sistemas de Informação** |
| --- | --- |

**Bruno Alison da Silva Melo - 262114100**

**Leonardo Kazunari Mada - 262113996**

**Gabriel Barbosa Salvador - 262114569**

**Atividade Avaliativa de Inteligência Artificial (1º Bimestre)**

**Trabalho de Desenvolvimento - 8puzzle**

Presidente Prudente – SP

**SUMÁRIO**

[**1.O que é o 8-puzzle 3**](#_heading=h.mcf620vyprqy)

[**2.Descrição da aplicação 3**](#_heading=h.ux9qyzhgdbbb)

[**3.Apresentação da interface gráfica (tela) 3**](#_heading=h.melbm8hqpeih)

[3.1.Painel Esquerdo 4](#_heading=h.p6ni3hbd98zm)

[3.1.1.Definição de Estado Inicial e Final 4](#_heading=h.1b4yxzrqpmuw)

[3.1.2.Seleção de algoritmo de busca 4](#_heading=h.chcs1rustp1q)

[3.1.3.Seleção de heurística do algoritmo 4](#_heading=h.o5wb48dc5s04)

[3.1.4.Seleção do tipo de visualização 5](#_heading=h.tq5bbf9n7cxb)

[3.1.5.Execução da busca 5](#_heading=h.1ec5fngaen66)

[3.2.Painel Direito 6](#_heading=h.6lme0kg6ixp)

[3.2.1.Relatório de resultados da busca 6](#_heading=h.q7pkznis9l16)

[3.2.2.Exibição do Estado Atual e Árvore de Passos 6](#_heading=h.lg40jhueag0w)

[**4.Lógica de processamento das buscas selecionadas 7**](#_heading=h.7pu8plaesek)

[4.1.Busca em Largura 7](#_heading=h.lljxdhxk94kj)

[4.2.Busca em profundidade 7](#_heading=h.qx8t0xqwttip)

[4.3.Busca A\* (A-Estrela) 8](#_heading=h.opel3jub611i)

[**5.Lógica de processamento das heurísticas selecionadas 8**](#_heading=h.wzin08pqifqs)

[5.1.Distância Manhattan 8](#_heading=h.ostc5jmokm8z)

[5.2.Peças Fora do Lugar 8](#_heading=h.4l8qtnsxs44x)

[**6.Comparativo de eficiência entre as buscas 9**](#_heading=h.vzws0z9hj1qv)

[6.1.Visualização dos resultados de comparativo 10](#_heading=h.my7zvxk7s093)

# 

# 1.O que é o 8-puzzle

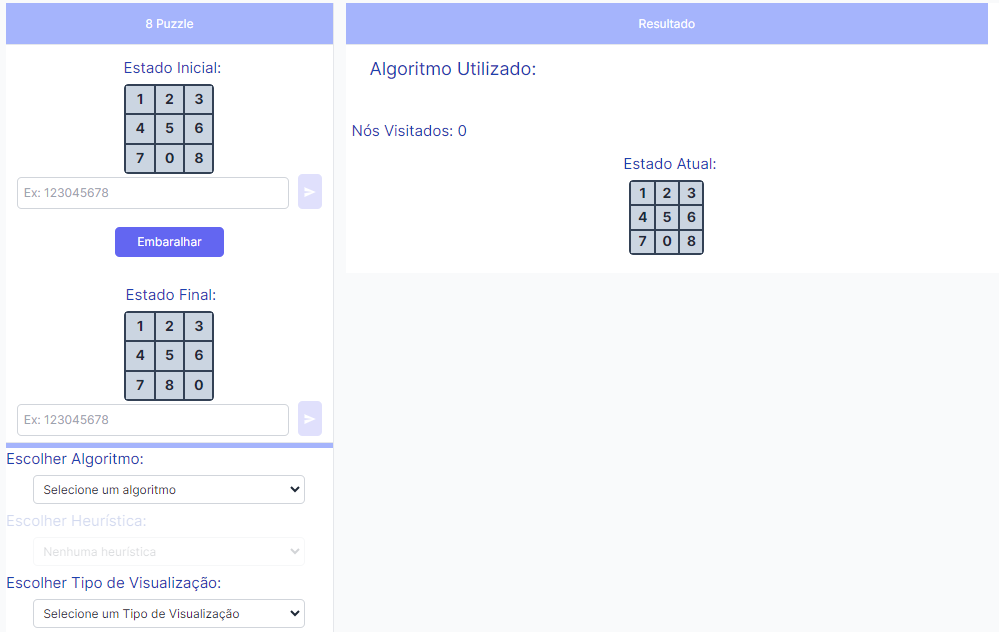
O 8-puzzle é um quebra-cabeça deslizante que consiste em um tabuleiro 3x3 com 8 peças numeradas de 1 a 8 e um espaço em branco (neste caso, representado por 0), permitindo que as peças sejam movidas para o espaço em branco. O objetivo é rearranjar as peças, movendo-as uma de cada vez para alcançar uma configuração específica, geralmente começando de uma configuração desordenada até chegar a uma configuração ordenada. O quebra-cabeça é um exemplo clássico de problemas de busca e é frequentemente utilizado como um problema de teste para algoritmos de inteligência artificial, como algoritmos de busca heurística.

# 

# 2.Descrição da aplicação

O trabalho deve permitir que o usuário embaralhe as peças de “Estado Inicial” e “Estado Final” do jogo, para que por meio de alguma busca e a forma de visualização da construção da solução selecionados pelo mesmo, o algoritmo consiga encontrar a sequência correta de passos e retornar os valores para a tela de forma síncrona e ordenada.

# 3.Apresentação da interface gráfica (tela)

A tela é dividida em 2 painéis.

## 3.1.Painel Esquerdo

No painel à esquerda se encontram as opções de configuração da situação problema, sendo elas: Definição de estados (Inicial e Final), seleção do algoritmo de busca, seleção de heurística (dependendo da escolha do algoritmo), seleção do tipo de visualização, opção de execução (iniciar a busca ou cancelar uma busca recorrente).

### 3.1.1.Definição de Estado Inicial e Final

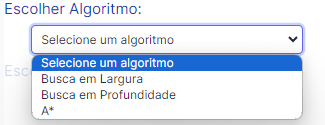
Nos campos de definição do puzzle, o usuário pode inserir um Estado Inicial e Final de sua preferência, optar pelos Estados Default pré-definidos pelos desenvolvedores da aplicação ou o embaralhamento realizado pela própria aplicação.

Por meio do campo input, cada respectivos 3 dígitos inseridos correspondem a uma linha da matriz (o software permite apenas a inserção dos números de 0 a 8, sem repetição) de modo que a sequência “012345678” represente:



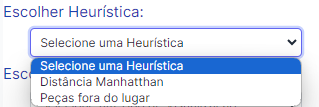
### 3.1.2.Seleção de algoritmo de busca

No campo de escolha de algoritmo, estão disponíveis as opções: Busca em Largura, Busca em Profundidade e A\* (A-Estrela). Cada opção exige um comportamento de busca de solução diferente, interferindo diretamente nos valores dos resultados de quantidade de passos (nós) visitados para obter a solução, o tempo gasto, tamanho do caminho da solução encontrada.



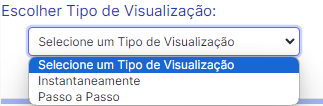
### 3.1.3.Seleção de heurística do algoritmo

Caso o usuário escolha a opção de algoritmo A\*, deve ser selecionada também uma heurística (Distância Manhatthan ou Peças fora do lugar), para que o algoritmo diferencie a prioridade de processamento de cada possibilidade.



### 3.1.4.Seleção do tipo de visualização

No campo de escolha do tipo de visualização, o usuário define a forma como quer a visualização dos nós (possibilidades ou visitados) retornados para a tela, optando pela visualização Instantânea ou acompanhar o Passo a Passo da busca.



Caso o usuário escolha a opção de Passo a Passo, serão habilitados os controles de voltar o passo e próximo passo, para acompanhar o processamento dos nós na busca pela solução.



### 3.1.5.Execução da busca

Com todas as opções de configuração necessárias selecionadas, o botão de busca de solução será habilitado, assim o usuário pode iniciar o processamento de busca da solução.



Ao iniciar, a opção de cancelar a busca de solução será habilitada, para caso o usuário deseje alterar configurações antes do retorno da busca atual (o processo descarta os dados processados até o momento).



## 3.2.Painel Direito

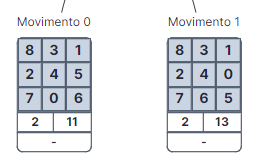
No painel à direita se encontra a visualização dos valores gerados pela busca da situação problema, sendo eles: o relatório contendo a quantidade de passos (nós) visitados para obter a solução, o tempo gasto e tamanho do caminho da solução encontrada, a exibição do puzzle de estado (nó atual) e a exibição/construção da árvore de nós.

### 3.2.1.Relatório de resultados da busca

O relatório conta com a apresentação dos resultados de: Tempo de solução (tempo decorrido do início ao fim da busca da solução), Quantidade de nós visitados (quantidade total de nós/passos visitados para buscar o caminho da solução), Quantidade de nós até a Solução (quantidade de nós que compõem o caminho da solução do estado inicial ao final).

### 3.2.2.Exibição do Estado Atual e Árvore de Passos

As exibições trazem ao usuário o Estado Atual do puzzle e a árvore com os passos realizados, contendo informações próprias do processamento em seus nós.



* Movimento N: número da possibilidade de movimento do nó atual.
* Matriz (área em azul): representa o estado do puzzle caso siga o movimento.
* Quadra à esquerda: nível na árvore (utilizado como função de custo).
* Quadra à direita: valor da função heurística (utilizado para algoritmos com critério de heurística)
* Quadro abaixo: ordem em que o estado foi visitado durante a busca.

Colorações de Nó/Estado:

* Azul: o estado foi armazenado, mas ainda não visitado.
* Amarelo: o estado compõe o caminho da solução.
* Vermelho: o estado é incapaz de trazer a solução.
* Verde: o estado é o estado buscado.

# 4.Lógica de processamento das buscas selecionadas

## 4.1.Busca em Largura

O algoritmo de busca em largura é uma técnica de busca não informada que explora todos os nós de um grafo de maneira sistemática e em largura, ou seja, expande primeiro todos os nós vizinhos do nível atual antes de expandir os nós do próximo nível. Ele é frequentemente usado para encontrar o caminho mais curto de um nó inicial para um nó objetivo em grafos sem pesos (onde todas as arestas têm o mesmo custo). O algoritmo é completo e sempre encontrará uma solução se ela existir, desde que o grafo seja finito. No entanto, sua complexidade de tempo e espaço pode ser alta, especialmente em grafos grandes ou com muitos nós, devido à necessidade de armazenar todos os nós em uma fila e à exploração exaustiva de todas as possibilidades antes de encontrar o caminho mais curto.

## 4.2.Busca em profundidade

O algoritmo de busca em profundidade (DFS - Depth-First Search) é outro método de busca não informada, geralmente usado para explorar grafos. Ao contrário do algoritmo de busca em largura, o DFS prioriza a exploração de um ramo do grafo o mais profundamente possível antes de retroceder. É importante notar que o DFS não garante a descoberta do caminho mais curto, ao contrário do algoritmo de busca em largura. Dependendo da estrutura do grafo, o DFS pode ficar preso em ramos infinitos ou explorar profundamente antes de encontrar uma solução, o que pode torná-lo ineficiente para certos problemas. No entanto, o DFS é útil em várias aplicações, como na busca de soluções para problemas de labirinto ou na travessia de árvores de busca em algoritmos como a busca em profundidade limitada e a busca em profundidade iterativa.

## 4.3.Busca A\* (A-Estrela)

O algoritmo A\* é uma combinação de busca heurística e busca de custo uniforme. Ele utiliza uma função de avaliação (ou heurística) para estimar o custo do melhor caminho entre um nó inicial e um nó objetivo. A\* mantém duas listas de nós: uma lista de nós a serem avaliados e uma lista de nós já avaliados. Ele expande primeiro os nós com menor custo total (custo do caminho percorrido até o nó mais a estimativa heurística do custo do caminho até o objetivo). Sendo considerado ótimo se a heurística utilizada for admissível (nunca superestima o custo para alcançar o objetivo) e consistente (a estimativa de custo de um nó é menor ou igual ao custo de chegar a qualquer nó sucessor mais a estimativa de custo do nó sucessor).

# 5.Lógica de processamento das heurísticas selecionadas

Ao optar por resolver o 8-puzzle usando A\*, o algoritmo precisa de uma maneira de estimar o quão longe o “Estado Atual” está do “Estado Final” a cada movimento.

## 5.1.Distância Manhattan

Para cada peça no tabuleiro, deve ser avaliada a distância que ela precisa percorrer para chegar à sua posição correta no tabuleiro final. Para isso, é calculada a distância horizontal e vertical entre a posição atual da peça e sua posição final desejada.

Por exemplo, se o número 3 está atualmente na posição (1,2), mas precisa estar na posição (0,1) no estado final, a distância horizontal é |1-0| = 1 e a distância vertical é |2-1| = 1. Portanto, a distância de Manhattan para o número 3 seria 1 + 1 = 2.

O cálculo deve ser feito para todas as peças, os resultados de todas as distâncias Manhattan devem ser somados para obter uma estimativa do quão longe o estado atual está do objetivo. Assim o algoritmo pode decidir qual movimento fazer a seguir, priorizando os movimentos que se aproximam do objetivo.

## 5.2.Peças Fora do Lugar

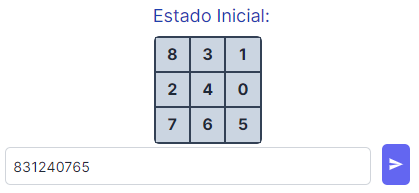
Para cada estado do tabuleiro, deve ser avaliada a quantidade de peças fora do lugar em relação ao tabuleiro final. Para isso, são contadas as peças que se encontram na posição adequada ao “Estado Final”, subtraindo as mesmas da quantidade total para encontrar a quantidade fora do lugar.

O cálculo deve ser feito para todas as peças, o resultado da contagem de peças fora do lugar estima ao algoritmo o próximo estado que mais se aproxima do objetivo.

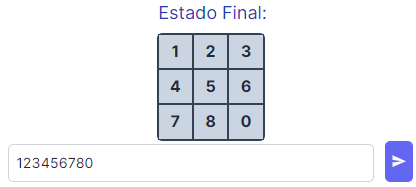
# 6.Comparativo de eficiência entre as buscas

Como exemplo para a comparação de eficiência das buscas serão apresentados os casos default, onde:

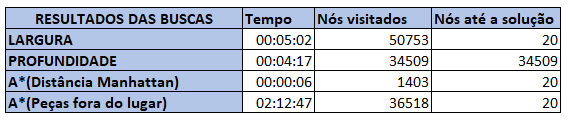
Estado Inicial = “831240765”



Estado Final = “123456780”



Após a realização das buscas os resultados obtidos foram:



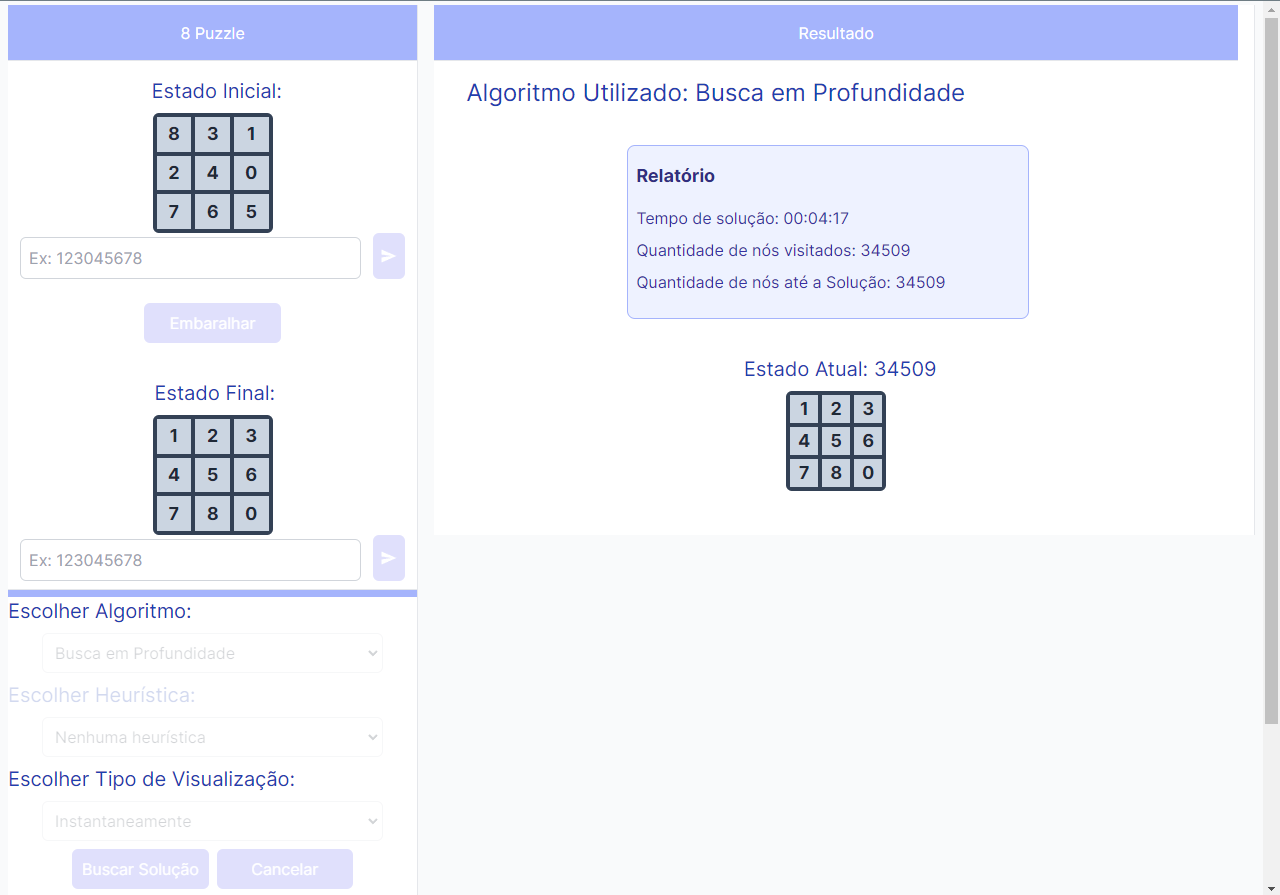
No comparativo default a busca A\* com heurística de Distância Manhattan apresentou os melhores resultados, passando por menos estados até encontrar o estado final.

## 6.1.Visualização dos resultados de comparativo

Busca em Largura:



Busca em Profundidade:



A\* (com heurística Distância de Manhattan):



A\* (com heurística Peças fora do lugar):

