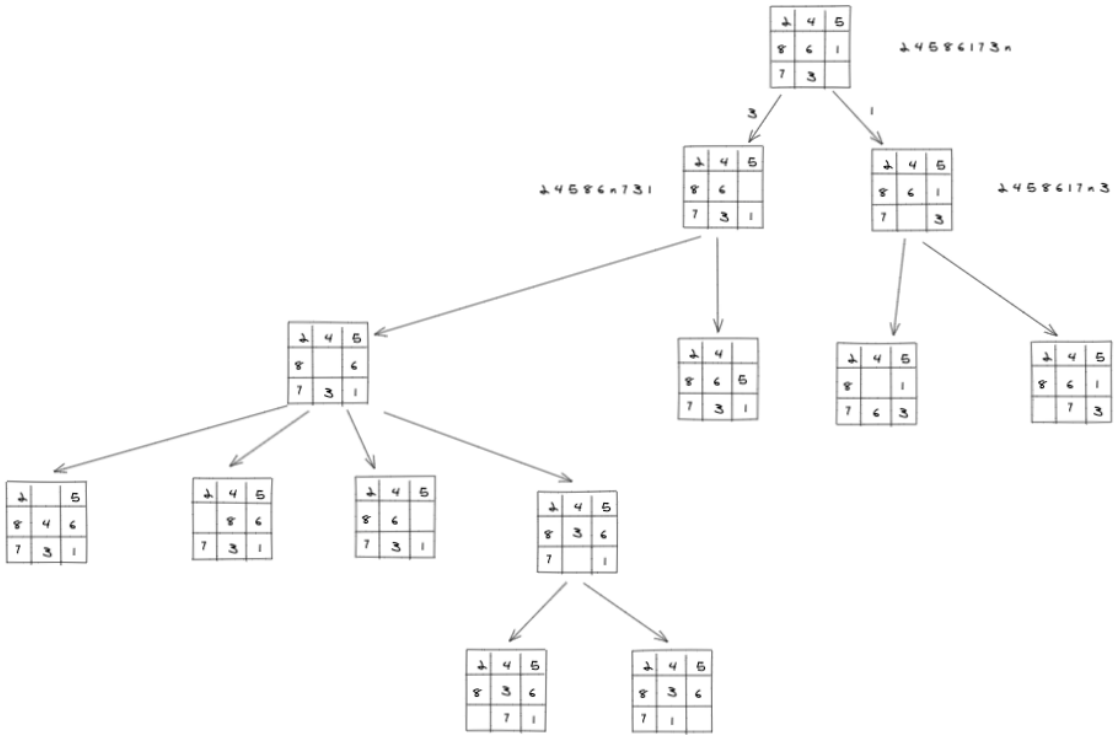
**1.Qual a representação (estrutura de dados) do estado**

A configuração dos estados foi feita através da representação de uma matriz em forma de lista, sendo cada index do array equivalente a uma posição da matriz e o uma posição NULL representando o espaço vazio.

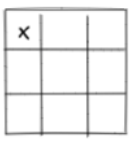

2 4 5 8 6 1 7 3 n

Exemplo:

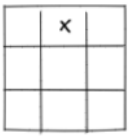


Para desenvolver a regra que valida se o movimento é válido ou não seguimos o seguinte processo:

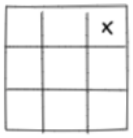
1. A partir dessa visualização foi possível identificar padrões entre os diferentes movimentos relativos a peça que está vazia:



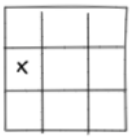

0 -> 1
0 -> 3



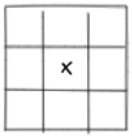

1 -> 0
1 -> 2
1 -> 4



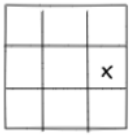

2 -> 1
2 -> 5



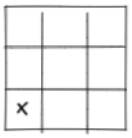

3 -> 0
3 -> 4
3 -> 6



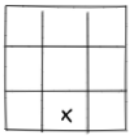

4 -> 1
4 -> 3
4 -> 5
4 -> 7



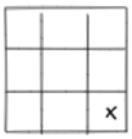

5 -> 2
5 -> 4
5 -> 8



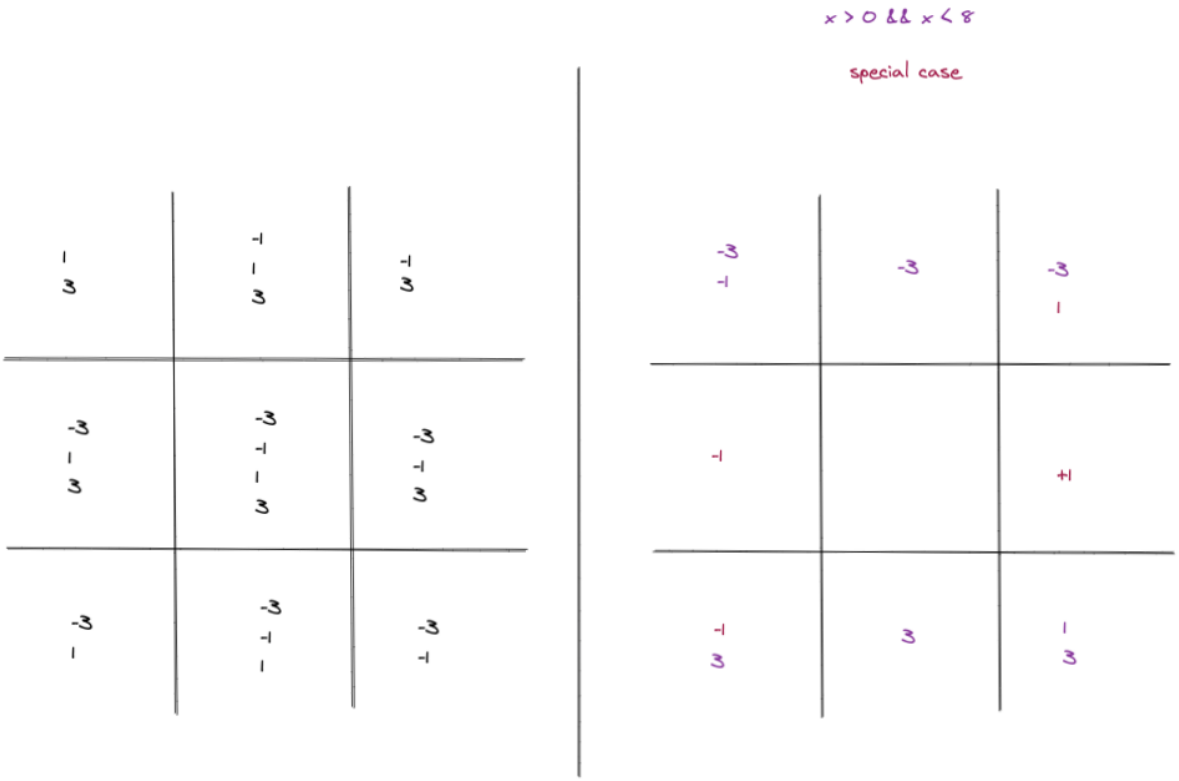

6 -> 3
6 -> 7




7 -> 4
7 -> 6
7 -> 8




8 -> 5
8 -> 7




A primeira descoberta foi que os movimentos possíveis variam entre os elementos do conjunto {-3, -1, 1, 3 }. Sendo que algumas posições possuem regras especiais que limitam os movimentos que seriam impossíveis de realizar no tabuleiro físico, são elas:

- Index + variante >= 0
- Index + variante < 9
- Se o index da posição vazia for 2 ou 5: a variante tem que ser diferente de 1
- Se o index da posição vazia for 3 ou 6: a variante tem que ser diferente de -1

Variante = elemento do conjunto que representa a movimentação de uma peça

**2.Qual a estrutura de dados para a fronteira e nodos fechados**

Nodos fechados: estrutura de lista (array)

Nodos abertos: lista ordenada pelo valor do estado que é equivalente ao número de peças fora da posição final.

**Exemplo de representação: listas**

• Estado inicial: [2,null,3,1,6,4,7,**0**,5]
• Estado objetivo: [0,1,2,3,4,5,6,7,null]
• exemplos de operadores
[a,b,c,d,e,f,g,h,0] → [a,b,c,d,e,f,g,0,h] p/ esquerda

[a,b,c,d,e,f,g,h,0] → [a,b,c,d,e,0,g,h,f] p/ cima

**3.Descrição da implementação (ideia geral e métodos relacionados) das heurísticas**

**Busca heurística**

A heurística se baseia no número de peças que estão fora da posição final.
Estado inicial: [0, null,2,3, 1,5,6,4,7]
Estado objetivo: [0,1,2,3,4,5,6,7,null]
Peças fora da posição objetivo: 3

**Busca pela melhor escolha**

Em cada etapa escolhemos o nó mais promissor gerado até o momento
Utiliza uma função de avaliação que retorna o custo de se chegar a uma solução (quanto menor melhor), método esse derivado do A*

**4.Como foi gerenciada a fronteira, verificações, quais etapas foram feitas ao adicionar um estado na fronteira (explicação das estratégias, respectivos métodos e possibilidades além do que foi implementado)**

O gerenciamento da fronteira foi a partir de uma lista ordenada pelo valor calculado através da heurística de cada estado. A cada inserção a lista é reordenada de forma crescente e toda remoção era feita na posição inicial da lista.

A inserção só ocorre quando o nodo não está contido na lista de nodos visitados e/ou na fronteira. Essa validação ocorre através da comparação entre estados através da sua respectiva representação em forma de string, evitando assim a comparação de referências entre os arrays.

Um dos pontos negativos da abordagem escolhida é de que a função que obtém o valor do estado se baseia no index e no valor de cada posição, sendo assim o algoritmo só consegue rodar corretamente com cada posição contendo um número.

**5.Quais os métodos principais e breve descrição do fluxo do algoritmo?**

1. O programa começa criando a estrutura de dados para lista de nodos abertos referente ao algoritmo a ser utilizado (Custo uniforme – lista ordenada pelo valor do estado, A* - lista ordenada pelo valor do estado + valor nível de profundidade).
2. Um nodo inicial é criado a partir do nodo objetivo (array de números de 0 a 7 mais uma posição NULL) de forma randômica e garantindo que será gerado um estado alcançável. Este nodo é inserido na estrutura de dadas criada no passo 1.
3. A cada segundo uma função é executada representando o próximo turno, nela são realizados os seguintes passos:
    a. Aumento no contador de turnos
    b. Remoção do elemento da estrutura de nodos abertos através do método *pop,* movendo a responsabilidade de regra de remoção da fronteira do escopo do Jogo para a classe SortedList.

c. Este nodo é inserido na lista de nodos visitados.
d. É feita a validação pra saber se o estado é o nodo objetivo, caso for o método retorna true, caso não ele continua.
e. A partir desse nodo são gerados os possíveis estados subsequentes sem levar em consideração nodos abertos ou visitados.
f. Esta lista de nodos é iterada agora validando se o nodo já foi visitado e se já não está na fronteira, caso não esteja em nenhuma das duas ele é adicionado a fronteira.

**6.Caso algum dos objetivos não tenha sido alcançado explique o que você faria VS o que foi feito e exatamente**

O algoritmo não alcançou a profundida de resolução esperada de no máximo 32 níveis, já casos em que a solução é encontrada com mais de até 150 níveis. Atribuímos essa falha a validação de nodos abertos/visitados que aparenta funcionar corretamente com poucos nodos, porém em casos reais acaba se mostrando não totalmente confiável.

No início da implementação não queríamos que o algoritmo dependesse do valor das posições para calcular a heurística, infelizmente não conseguimos implementar uma outra forma.

A natureza de execução do algoritmo se mostrou muito difícil de debuggar e encontrar erros, o que atrapalhou o desenvolvimento como um todo. Uma implementação que possibilita visualizar os nodos abertos e visitado foi criada pra auxiliar no desenvolvimento.

O objetivo de profundidade máxima não foi alcançado.