

Implementação

Para realizarmos a implementação do código, utilizaremos múltiplas funções para facilitar na economia de linhas e memória RAM, além disso traremos uma explicação de cada atividade dentro do código. Ainda, o trabalho foi desenvolvido dentro da plataforma Apache Netbeans, pois existe uma familiaridade com o programa e compatibilidade com o que é apresentado na sala de aula.

1. Importações

Primeiro, para darmos início a implementação do nosso código, importaremos duas classes, a **Random** e a **Scanner**, utilizando o método **import java.util** da seguinte maneira:

```
1 import java.util.Random;
2 import java.util.Scanner;
```

- Importar a **classe Random**: que é usada para gerar números aleatórios e servirá para embaralhar inicialmente os números do tabuleiro
- Importar a **classe Scanner**: que permite ler a entrada do usuário.
- Para importar a **classe Random** foi necessário uma pesquisa na internet de como funciona sua implementação, e será usada para mostrar de qual maneira em que deve ser feito o embaralhamento dos números no tabuleiro dentro da linguagem Java.
- A **classe Scanner** já foi apresentada em sala de aula e é utilizada para receber os dados de entrada do usuário.

2. Declaração de Classe e Variáveis

Posteriormente, iremos iniciar a classe **RachaCuca** e iremos criar as variáveis necessárias dentro do nosso código da seguinte forma e com devida explicação:

```

4 public class RachaCuca {
    private static final int tamanho = 3;
    private int[][] tabuleiro;
7 private int linhavazia, colunavazia;

```

- **private static final int tamanho = 3:** Usado para definir o tamanho do tabuleiro como 3x3.
- **private int[][] tabuleiro:** Declarar uma matriz bidimensional para representar o tabuleiro.
- **private int linhavazia, colunavazia:** Variáveis para armazenar a posição da célula vazia.

Após isso, iremos criar um método de inicialização do tabuleiro a fim de ordenar os tamanhos e criando uma nova função, chamada de **public RachaCuca** e logo após o método **inicio ()**.

```

8
9 public RachaCuca () {
10     tabuleiro = new int[tamanho][tamanho];
11     inicio();
12 }

```

Logo após, criamos uma nova função para preenchimento do tabuleiro de forma inicial, utilizando laços de repetição com as variáveis **i e j**, de forma também que o último elemento do tabuleiro seja vazio para início do jogo, da seguinte maneira:

```

14 private void inicio() {
15     int num = 1;
16     for (int i = 0; i < tamanho; i++) {
17         for (int j = 0; j < tamanho; j++) {
18             if (num < tamanho * tamanho) {
19                 tabuleiro[i][j] = num++;
20             } else {
21                 tabuleiro[i][j] = 0; // Espaço vazio
22                 linhavazia = i;
23                 colunavazia = j;
24             }

```

- **private void início ()**: Método que irá preencher o tabuleiro com números de 1 a 8 e definir a posição do espaço vazio como 0.
- O loop preenche a matriz e, quando chega ao último espaço, define-o como vazio.

3. Embaralhamento do tabuleiro

Na sequência, temos uma nova função, dessa vez para embaralhar o tabuleiro inicialmente de forma aleatória com base na escolha de dificuldade do usuário, utilizando novamente um laço de repetição para preenchimento e movimentação das peças.

Além disso, também utilizamos duas funções, **a Random e switch**, para utilizar o switch tivemos que efetuar pesquisas na internet para entender sobre economia de linhas dentro das escolhas e posteriormente seguida de um **break (pare)** caso a situação seja positiva. Na função switch, aprendemos que é uma ferramenta utilizada para tomar decisões com base no valor de uma expressão.

```

29 private void embaralhartabuleiro (int moves) {
30     Random rand = new Random();
31     for (int i = 0; i < moves; i++) {
32         int direcao = rand.nextInt(4);
33         switch (direcao) {
34             case 0: move(linhavazia - 1, colunavazia); break; // Cima
35             case 1: move(linhavazia + 1, colunavazia); break; // Baixo
36             case 2: move(linhavazia, colunavazia - 1); break; // Esquerda
37             case 3: move(linhavazia, colunavazia + 1); break; // Direita
38         }
39     }

```

- **private void embaralhartabuleiro (int moves)**: Método que irá embaralhar o tabuleiro movendo o espaço vazio em direções aleatórias. Ele é declarado como private, o que significa que ele só pode ser chamado dentro da própria classe RachaCuca. Isso é útil para encapsular a lógica de embaralhamento e evitar que outras partes do código interfiram diretamente nesse processo.
- O número de movimentos será determinado pelo **parâmetro moves**.
- **Random rand = new Random ()**; criamos uma instância da classe Random, que é usada para gerar números aleatórios.
- **for (int i = 0; i < moves; i++)**: é um loop que se repete **moves** vezes. A cada iteração, uma nova direção de movimento é escolhida aleatoriamente.

- **int direction = rand.nextInt(4);** gera um número aleatório entre 0 e 3. Esse número representa uma direção:
 - 0: Cima
 - 1: Baixo
 - 2: Esquerda
 - 3: Direita
- O switch é usado para decidir qual movimento executar com base no número gerado:
- **Cima:** move(linhavazia- 1, colunavazia) tenta mover a peça acima do espaço vazio.
- **Baixo:** move(linhavazia + 1, colunavazia) tenta mover a peça abaixo do espaço vazio.
- **Esquerda:** move(linhavazia, colunavazia - 1) tenta mover a peça à esquerda do espaço vazio.
- **Direita:** move(linhavazia, colunavazia + 1) tenta mover a peça à direita do espaço vazio.
- O método move(int linhavazia, int colunavazia) é chamado para realizar a movimentação. Esse método verifica se o movimento é válido (ou seja, se a peça está adjacente ao espaço vazio) e, se for, executa a movimentação.

4. Movimentação de peças

Em sequência, temos a utilização de um método para realizarmos a movimentação das peças escolhidas pelo jogador, além de realização de validação se o movimento pode ser executado e se será realizado para uma casa que se encontra vazia, também é realizada uma verificação sobre a nova posição das peças dentro do tabuleiro. Segue imagem do código e paragrafação com explicações:

```

42 private boolean move(int novalinha, int novacoluna) {
43     if (movimentovalido(novalinha, novacoluna)) {
44         tabuleiro[linhavazia][colunavazia] = tabuleiro[novalinha][novacoluna];
45         tabuleiro[novalinha][novacoluna] = 0;
46         linhavazia = novalinha;
47         colunavazia = novacoluna;
48         return true;
49     }
50     return false;
51 }

```

- **private boolean move(int novalinha, int novacoluna):** Este método irá receber duas variáveis inteiras, **novalinha e novacoluna**, que representam a nova posição para onde a peça deve ser movida. O método retornará um valor booleano (true ou false), indicando se o movimento foi bem-sucedido.
- **if (movimentovalido(novalinha, novacoluna)):** Aqui, o método chama movimentovalido, que verifica se a nova posição (novalinha, novacoluna) é válida. Isso significa que a posição deve estar dentro dos limites do tabuleiro (0 a 2, já que o tamanho é 3x3). Se o movimento for válido, o código dentro do bloco if será executado.
- **Tabuleiro [linhavazia][colunavazia] = board[novalinha][novacoluna];:** Esta linha irá mover a peça da posição (novalinha, novacoluna) para a posição vazia (linhavazia, colunavazia). O valor da peça que estava na nova posição é copiado para a posição vazia, efetivamente “movendo” a peça.
- **board[novalinha][novacoluna] = 0;:** Após mover a peça, esta linha define a nova posição (novalinha, novacoluna) como vazia, representada pelo valor 0. Isso é importante para manter a lógica do jogo, onde sempre deve haver um espaço vazio.
- **linhavazia = novalinha; e colunavazia= novacoluna;:** Essas linhas atualizam as variáveis linhavazia e colunavazia para refletir a nova posição do espaço vazio.

Isso será crucial para que o jogo saiba onde o espaço vazio está após o movimento.

- **return true;** Se o movimento foi realizado com sucesso, o método retorna true, indicando que a operação foi bem-sucedida.
- **return false;** Se o movimento não for válido (ou seja, se **movimentovalido** retornar false), o método retorna false, indicando que o movimento não pôde ser realizado.

5. Validação de movimento

Na sequência, continuamos a apresentação do código através da validação dos movimentos escolhidos pelo jogador, com ele verificamos se a lógica de movimentação das peças será válida dentro do tabuleiro, deixando sempre uma posição vazia. Além disso, será verificado se o tabuleiro ficou organizado com as peças em ordem crescente, que indicará uma resolução do tabuleiro, segue imagem e explicação.

```
53 private boolean movimentovalido(int linha, int coluna) {
54     return linha >= 0 && linha < tamanho && coluna >= 0 && coluna < tamanho;
55 }
56
57 private boolean resolvido() {
58     int num = 1;
59     for (int i = 0; i < tamanho; i++) {
60         for (int j = 0; j < tamanho; j++) {
61             if (i == tamanho - 1 && j == tamanho - 1) {
62                 return tabuleiro[i][j] == 0; // Última posição deve estar vazia
63             }
64             if (tabuleiro[i][j] != num++) {
65                 return false;
66             }
67         }
68     }
69 }
```

- **private boolean movimentovalido(int linha, int coluna):** Este método será responsável por verificar se uma posição (definida por linha e coluna)

é válida dentro dos limites do tabuleiro. Ele retorna um valor booleano (true ou false).

- **return linha >= 0 && linha < tamanho && coluna >= 0 && coluna < tamanho;** Esta linha contém a lógica que determina se a posição é válida:
 - linha >= 0: Verifica se a linha (linha) é maior ou igual a 0. Isso garante que não estamos tentando acessar uma linha negativa.
 - linha < tamanho: Verifica se a linha é menor que tamanho (que é 3). Isso garante que não estamos tentando acessar uma linha fora do limite superior do tabuleiro.
 - coluna >= 0: Verifica se a coluna (coluna) é maior ou igual a 0, garantindo que não estamos acessando uma coluna negativa.
 - coluna < tamanho: Verifica se a coluna é menor que tamanho, garantindo que não estamos acessando uma coluna fora do limite superior.
 - **private boolean resolvido ():** Este método irá verificar se o tabuleiro está na configuração correta para ser considerado “resolvido”. Ele retorna true se o quebra-cabeça estiver resolvido e false caso contrário.
- int num = 1;** Inicializa uma variável num com o valor 1. Esta variável será usada para verificar se os números no tabuleiro estão na ordem correta.
- **for (int i = 0; i < tamanho; i++):** Inicia um loop que percorre as linhas do tabuleiro.
 - **for (int j = 0; j < tamanho; j++):** Inicia um loop aninhado que percorre as colunas do tabuleiro.
 - **if (i == tamanho - 1 && j == tamanho - 1):** Verifica se estamos na última posição do tabuleiro (canto inferior direito).
 - **return tabuleiro[i][j] == 0;** Se estivermos na última posição, verifica se ela é igual a 0 (o espaço vazio). Se for, retorna true, indicando que a última posição está correta.
 - **if (tabuleiro[i][j] != num++):** Para todas as outras posições, verifica se o valor na posição atual do tabuleiro (tabuleiro[i][j]) é igual ao número esperado (num).
 - Se não for igual, retorna false, indicando que o tabuleiro não está na ordem correta.
 - O operador num++ incrementa num após a comparação, preparando-o para a próxima posição.

- **return true;;** Se todas as posições foram verificadas e estão corretas, o método retorna true, indicando que o quebra-cabeça está resolvido.

5. Imprimindo tabuleiro

No próximo passo, iremos realizar um trabalho a fim de demonstrar a impressão do tabuleiro para o usuário depois das escolhas realizadas.

```
72 public void mostrartabuleiro() {  
73     for (int i = 0; i < tamanho; i++) {  
74         for (int j = 0; j < tamanho; j++) {  
75             if (tabuleiro[i][j] == 0) {  
76                 System.out.print("  ");  
77             } else {  
78                 System.out.printf("%2d ", tabuleiro[i][j]);  
79             }  
80         }  
81         System.out.println();  
82     }  
83 }
```

- **public void mostrar tabuleiro ():** Este é um método público que não retorna nenhum valor (void). Ele é chamado para imprimir o tabuleiro do jogo.
- **for (int i = 0; i < tamanho; i++):** Este loop irá percorrer as linhas do tabuleiro. A variável i representa o índice da linha atual, começando de 0 até tamanho - 1 (que é 2, já que tamanho é 3).
- **for (int j = 0; j < tamanho; j++):** Este é um loop aninhado que percorre as colunas do tabuleiro. A variável j representa o índice da coluna atual, também começando de 0 até tamanho - 1.
- **if (tabuleiro[i][j] == 0):** Aqui, o código verifica se a posição atual do tabuleiro (tabuleiro[i][j]) é igual a 0, que representa o espaço vazio
- **System.out.print(" ");** Se for um espaço vazio, imprime três espaços em branco. Isso ajuda a manter o alinhamento visual do tabuleiro, já que não há número a ser exibido.

- **else:** Se a posição não for vazia (ou seja, contém um número):
- **System.out.printf("%2d ", tabuleiro[i][j]);** Esta linha imprime o número na posição atual. O formato %2d garante que o número ocupe pelo menos dois espaços, alinhando os números corretamente no tabuleiro. O espaço após %2d adiciona um espaço extra entre os números.
- **System.out.println();** Após imprimir todos os números de uma linha, esta linha imprime uma nova linha. Isso move o cursor para a próxima linha do console, preparando-o para imprimir a próxima linha do tabuleiro.

6. Começando a jogar

No próximo passo, começamos a jogar e interagir dentro da solução, desde escolhendo cada peça queremos movimentar e como ela afeta o nosso jogo. Além disso, também trazemos uma mensagem caso a solução seja resolvida como também caso o movimento escolhido seja inválido. Segue explicação:

```

85 public void startGame() {
86     Scanner scanner = new Scanner(System.in);
87     while (true) {
88         mostrartabuleiro();
89         if (resolvido()) {
90             System.out.println("Parabens! Voce completou o quebra-cabeca! ");
91             break;
92         }
93         System.out.print("Escolha uma peça para mover (ou 0 para sair): ");
94         int choice = scanner.nextInt();
95         if (choice == 0) {
96             System.out.println("Obrigado por jogar! ");
97             break;
98         }
99         if (!makeMove(choice)) {
100             System.out.println("Movimento invalido! Tente novamente.");
101         }
102     }
103     scanner.close();

```

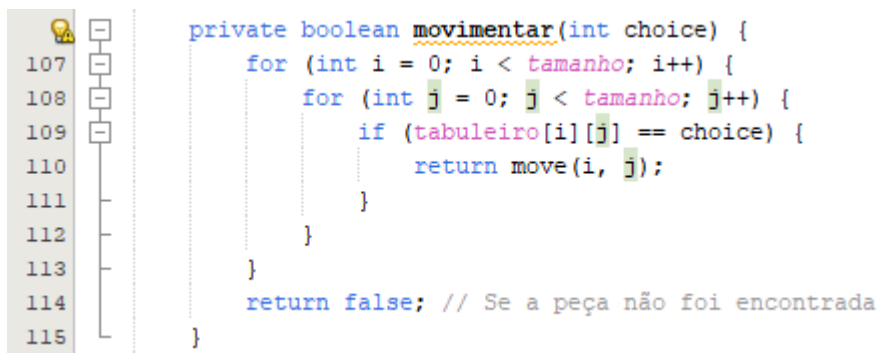
- **public void startGame():** Este é um método público que não retorna nenhum valor (void). Ele inicia o loop principal do jogo, permitindo que o jogador interaja com o tabuleiro.

- **Scanner scanner = new Scanner(System.in);**: Cria um objeto Scanner que será usado para ler a entrada do usuário a partir do console. Isso permite que o jogador insira suas escolhas.
- **while (true):** Inicia um loop infinito que continuará até que uma condição de saída seja atendida (usando break).
- **mostrartabuleiro();**: Chama o método mostrartabuleiro, que exibe o estado atual do tabuleiro no console. Isso permite que o jogador veja a configuração atual das peças.
- **if (resolvido()):** Verifica se o quebra-cabeça foi resolvido chamando o método resolvido.
- **System.out.println("Parabéns! Voce completou o quebra-cabeça");**: Se o quebra-cabeça estiver resolvido, imprime uma mensagem de congratulação.
- **break;**: Sai do loop, encerrando o jogo.
- **System.out.print("Escolha uma peça para mover (ou 0 para sair): ");**: Exibe uma mensagem solicitando que o jogador escolha uma peça para mover. O jogador pode inserir o número da peça ou 0 para sair do jogo.
- **int choice = scanner.nextInt();**: Lê a entrada do usuário e armazena o valor na variável choice. O jogador deve inserir um número correspondente à peça que deseja mover.
- **if (choice == 0):** Verifica se o jogador escolheu sair do jogo.
- **System.out.println("Obrigado por jogar! ");**: Imprime uma mensagem informando que o jogo está sendo encerrado.
- **break;**: Sai do loop, encerrando o jogo.
- **if (!makeMove(choice)):** Tenta mover a peça correspondente à escolha do jogador chamando o método makeMove.

- Se **makeMove** retornar false (indicando que o movimento não foi válido):
- **System.out.println("Movimento inválido! Tente novamente.");**:: Imprime uma mensagem informando que o movimento não é válido e solicita que o jogador tente novamente.
- **scanner.close();**:: Fecha o objeto Scanner após o término do jogo. Isso é uma boa prática para liberar recursos.

7. Movimentando as peças

Adiante, temos laços de verificação utilizados para avaliar se as movimentas realizadas são válidas, é um método privado usado apenas nesse momento do código.



```

107 private boolean movimentar(int choice) {
108     for (int i = 0; i < tamanho; i++) {
109         for (int j = 0; j < tamanho; j++) {
110             if (tabuleiro[i][j] == choice) {
111                 return move(i, j);
112             }
113         }
114     }
115     return false; // Se a peça não foi encontrada

```

- **private boolean movimentar(int choice):** Este é um método privado que recebe um parâmetro choice(escolha), que representa o número da peça que o jogador deseja mover. O método retorna um valor booleano (true ou false), indicando se o movimento foi bem-sucedido.
- **for (int i = 0; i < tamanho; i++):** Este loop percorre as linhas do tabuleiro. A variável i representa o índice da linha atual, começando de 0 até tamanho- 1 (que é 2, já que tamanho é 3).

- **for (int j = 0; j < tamanho; j++):** Este é um loop aninhado que irá percorrer as colunas do tabuleiro. A variável `j` representa o índice da coluna atual, também começando de 0 até tamanho- 1.
- **if (tabuleiro[i][j] == choice):** Aqui, o código verifica se a peça na posição atual do tabuleiro (`tabuleiro [i][j]`) é igual à escolha do jogador (`choice`). Se `for`, significa que o jogador escolheu uma peça válida para mover.
- **return move(i, j);:** Se a peça correspondente à escolha do jogador for encontrada, o método chama `move(i, j)`, passando as coordenadas da peça. O método `move` tentará mover a peça para a posição do espaço vazio. O resultado do movimento (se foi bem-sucedido ou não) é retornado diretamente.
- **}:** Este fechamento de chaves indica o fim do loop aninhado. Se a peça não for encontrada em nenhuma das posições do tabuleiro, o código continua.
- **return false;:** Se o loop terminar e nenhuma peça correspondente à escolha do jogador for encontrada, o método retorna `false`. Isso indica que o movimento não pôde ser realizado porque a peça não existe no tabuleiro.

8. Movimentando as peças

Então, esse é o nosso método `main` do código, nele estão inseridas grande parte das visões apresentadas ao usuário, tais como opções de escolha da dificuldade de jogo, mensagem de erro e término de jogo. Esse momento foi separado em 3 imagens diferentes, a fim de melhor visualização dentro da documentação.

```

117 public static void main(String[] args) {
118     RachaCuca game = new RachaCuca();
119     Scanner scanner = new Scanner(System.in);
120     while (true) {
121         System.out.println("Menu:");
122         System.out.println("0 - Sair");
123         System.out.println("1 - Novo Jogo");
124         System.out.println("2 - Instrucoes");
125         int option = scanner.nextInt();
126         if (option == 0) {
127             System.out.println("Jogo encerrado! Obrigado por jogar!");
128             break;
129         } else if (option == 1) {
130             System.out.print("Escolha o nivel de dificuldade (1 - Facil, "
131                             + "2 - Medio, 3 - Dificil): ");
132             int level = scanner.nextInt();
133             int moves = 20; // Default para fácil
134             if (level == 2) moves = 40;
135             else if (level == 3) moves = 80;
136             game.embaralhartabuleiro(moves);
137             game.startGame();
138         } else if (option == 2) {
139             mostrainstrucoes();
140         } else {
141             System.out.println("Opcao invalida! Tente novamente.");
142         }
143     }
144     scanner.close();
145 }

```

- **public static void main(String[] args):** Este é o método principal do nosso código e que será executado quando o programa é iniciado. É um método público e estático, o que significa que pode ser chamado sem criar uma instância da classe. O parâmetro String[] args permite que argumentos sejam passados para o programa via linha de comando, embora não sejam utilizados neste código.
- **RachaCuca game = new RachaCuca();:** Cria uma nova instância do jogo RachaCuca. Isso inicializa o tabuleiro e prepara o jogo para ser jogado.
- **Scanner scanner = new Scanner(System.in);** Cria um objeto Scanner para ler a entrada do usuário a partir do console. Isso permite que o jogador insira suas escolhas.
- **while (true):** Inicia um loop infinito que continuará até que uma condição de saída seja atendida (usando break).

- **System.out.println("Menu:");** Imprime o cabeçalho do menu.
- As próximas três linhas exibem as opções disponíveis para o jogador:
 - 0 - Sair: Para encerrar o jogo.
 - 1 - Novo Jogo: Para iniciar um novo jogo.
 - 2 - Instruções: Para ver as instruções do jogo.
- **int option = scanner.nextInt();** Lê a entrada do usuário e armazena o valor na variável option. O jogador deve inserir um número correspondente à opção desejada.
- **if (option == 0):** Verifica se o jogador escolheu sair do jogo.
- **System.out.println("Jogo encerrado! Obrigado por jogar!");** Imprime uma mensagem informando que o jogo está sendo encerrado.
- **break;** Sai do loop, encerrando o programa.
- **else if (option == 1):** Se o jogador escolher iniciar um novo jogo:
- **System.out.print("Escolha o nível de dificuldade (1 - Facil, 2 - Medio, 3 - Dificil): ");** Solicita que o jogador escolha o nível de dificuldade.
- **int level = scanner.nextInt();** Lê a escolha do nível de dificuldade.
- **int moves = 20;** // Default para fácil: Define o número padrão de movimentos para o nível fácil.
- **if (level == 2) moves = 40;** Se o nível médio for escolhido, define 40 movimentos.
- **else if (level == 3) moves = 80;** Se o nível difícil for escolhido, define 80 movimentos.

- **game.embaralhartabuleiro(moves);:** Chama o método `embaralhartabuleiro` para embaralhar o tabuleiro com o número de movimentos especificado.
- **game.startGame();:** Inicia o jogo chamando o método `startGame`.
- **else if (option == 2):** Se o jogador escolher ver as instruções: **mostrainstrucoes();:** Chama o método `mostrainstrucoes`, que exibe as regras do jogo.
- **else:** Se o jogador inserir uma opção que não é válida (não é 0, 1 ou 2):
- **System.out.println("Opcao invalida! Tente novamente.");:** Imprime uma mensagem informando que a opção é inválida e solicita que o jogador tente novamente.
- **scanner.close();:** Fecha o objeto `Scanner` após o término do jogo. Isso é uma boa prática para liberar recursos.

9. Método de instruções

Ademais, determinamos um método para representarmos como devem ser feitas as instruções caso o jogador escolha essa opção, também foi utilizada uma quebra de linha para melhor representação visual do código. Nesse sentido, ela é representada da seguinte maneira com paragrafação com marcadores:

```

149 private static void mostrarinstrucoes() {
150     System.out.println("Instrucoes:");
151     System.out.println("1. Mova as pecas adjacentes ao espaco vazio.");
152     System.out.println("2. O objetivo e ordenar as pecas de 1 a 8, "
153         + "deixando o espaco vazio no canto inferior direito.");
154     System.out.println("3. Digite o numero da peca que deseja mover ou 0 para sair.");
155 }
156
157

```

- **private static void mostrarinstrucoes():** Este é um método privado e estático, o que significa que ele pode ser chamado sem a necessidade de criar uma instância da classe RachaCuca. O método não retorna nenhum valor (void). Ele é usado para exibir as instruções do jogo.
- **System.out.println("Instrucoes:");**: Esta linha irá imprimir o título “Instruções:” para o usuário. É uma introdução que informa ao jogador que as instruções do jogo estão prestes a ser apresentadas.
- **System.out.println("1. Mova as pecas adjacentes ao espaco vazio.");**: Esta linha fornece a primeira instrução, explicando que o jogador pode mover as peças que estão adjacentes ao espaço vazio. Isso é fundamental para a mecânica do jogo, pois o jogador deve entender que só pode mover peças que estão ao lado do espaço vazio.
- **System.out.println("2. O objetivo é ordenar as peças de 1 a 8, deixando o espaco vazio no canto inferior direito.");**: Esta linha explica o objetivo do jogo. O jogador deve organizar as peças numeradas de 1 a 8 em ordem crescente, com o espaço vazio localizado na última posição (canto inferior direito). Essa informação é crucial para que o jogador saiba qual é a meta do jogo.
- **System.out.println("3. Digite o número da peca que deseja mover ou 0 para sair.");**: Esta linha orienta o jogador sobre como fazer um movimento. O jogador deve digitar o número da peça que deseja mover. Além disso, informa que o jogador pode digitar 0 para sair do jogo, oferecendo uma opção de saída clara.