Animação de Movimentação no Labirinto em Java

Técnicas de Programação Avançada - Prof. Mateus Conrad B. da Costa

Bacharelado em Sistemas de Informação - Ifes - Campus Serra

October 8, 2024

Introdução

- O código implementa uma animação de movimentação em um labirinto usando Java.
- Utiliza um grid de ‘JPanel‘ para representar o labirinto, onde:
  - 0 indica paredes.
  - 1 indica passagens.
- O jogador se move aleatoriamente pelo labirinto.

Classe Labirinto

- A classe ‘Labirinto‘ define o labirinto e gerencia a animação de movimentação.
- Declarações principais:

```
private Random random = new Random();
private JPanel[][] cells;
private int playerRow, playerCol;
private int[][] labirinto;
```

- Usa ‘Random‘ para escolher as direções do jogador.

Construtor Labirinto

- O construtor recebe uma matriz de inteiros representando o labirinto.
- Chamadas para inicialização da interface e animação:

```
public Labirinto(int[][] labirinto) {
    this.labirinto = labirinto;
    this.cells = new JPanel[labirinto.length][
    initUI();
    iniciarAnimacao();
}
```

- ‘initUI()‘ configura o layout gráfico, e ‘iniciarAnimacao()‘ inicia o movimento.

Configuração da Interface

- O método ‘initUI()‘ cria a janela e configura os componentes gráficos (‘JPanel‘).
- A janela usa um ‘GridLayout‘ com base no tamanho da matriz do labirinto.

```
setLayout(new GridLayout(labirinto.length,
labirinto[0].length));
```

- As paredes são pretas e as passagens são brancas:

```
if (labirinto[i][j] == 0) {
    cells[i][j].setBackground(Color.BLACK);
} else {
    cells[i][j].setBackground(Color.WHITE);
}
```

Posição Inicial do Jogador

- A posição inicial do "jogador" pode ser (0,0) ou aleatória, mas garantidamente em uma passagem.

```
do {
    playerRow = random.nextInt(
    labirinto.length);
    playerCol = random.nextInt(
    labirinto[0].length);
} while (labirinto[playerRow][playerCol] == 0);
```

- A célula do jogador é colorida de vermelho:

```
cells[playerRow][playerCol].setBackground(
Color.RED);
```

Início da Animação

- O método ‘iniciarAnimacao()‘ usa um ‘Timer‘ para mover o jogador a cada 500ms.

```
Timer timer = new Timer(500, e ->
moverJogador());
timer.start();
```

- A função ‘moverJogador()‘ será responsável pela movimentação aleatória.

Movimentação Aleatória

- O método ‘moverJogador()‘ escolhe uma direção aleatória (cima, baixo, esquerda, direita):

```
int direcao = random.nextInt(4);
switch (direcao) {
 case 0: novaLinha = playerRow - 1; break;
 case 1: novaLinha = playerRow + 1; break;
 case 2: novaColuna = playerCol - 1; break;
 case 3: novaColuna = playerCol + 1; break;
}
```

Verificação de Limites e Movimentação

- Antes de mover o jogador, verificamos se a nova posição é válida:

```
if (novaLinha >= 0 && novaLinha <
    labirinto.length &&
    novaColuna >= 0 && novaColuna <
    labirinto[0].length &&
    labirinto[novaLinha][novaColuna] == 1) {
        cells[playerRow][playerCol].
        setBackground(Color.WHITE);
        playerRow = novaLinha;
        playerCol = novaColuna;
        cells[playerRow][playerCol].
        setBackground(Color.RED);
}
```

Método Criar Labirinto

- O método ‘criarLabirinto()‘ gera uma matriz que representa o labirinto:

```
public static int[][] criarLabirinto() {
    return new int[][]{
        {1, 0, 0, 0, 0, 0, 0},
        {0, 1, 1, 0, 1, 1, 0},
        ...
        {0, 0, 0, 0, 0, 0, 1}
    };
}
```

Método Principal

- No método principal, o labirinto é criado e a animação é iniciada:

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
     Labirinto lab =
     new Labirinto(criarLabirinto());
    });
}
```

- Usa ‘SwingUtilities‘ para garantir que a GUI seja manipulada na thread correta.