Vitor Neuenschwander Mendes da Silva

Pilhas e Filas
Torre de Hanói

Belo Horizonte, Brasil

2024

1 Introdução

Implemetação visual do Jogo de Hanoi

- O objetivo do projeto era a implementação visual do jogo **Torre de Hanoi** no Processing por meio da funcionalidade **Stack** (Pilha) do pacote java.util. O jogo consiste em três estacas de mesmo tamaho e um número *n* de discos com diferentes larguras posisicionados em ordem crescente de acordo com as larguras na primeira torre, o objetivo final do jogo é posicionar todos esses discos na mesma ordem na terceira torre.
- No nosso caso usamos somente 5 discos com as seguintes modificações:
  - O jogo se inicia com 5 discos de tamanho aleátorio somente na 1ª torre
  - Somente o último disco pode ser removido de uma torre
  - Um disco maior não pode ser posicionado em cima de um menor
- Dessa forma o usuário deve clicar na torre que deseja remover o disco e em seguinte na torre destino.

2 Desenvolvimento

- **Arquivo Main**: Onde contruí as torres e os discos nelas contidos, e a partir disso a lógica de contrução, ordenação e interação do usuário com o programa.
  - **Variáveis**:
    * `Stack<Disco>[] torres = new Stack[3];`
      `//Representa as 3 torres com um Stack de discos em cada.`
    * `Stack<Float> estacaX = new Stack<>();`
      `//armazena Xpos das torres`
    * `LinkedList<Integer> sizes = new LinkedList<Integer>();`
      `//armazena os 5 possíveis tamanhos do disco`
    * `int[] swap = new int[2];`
      `//armazena torre origem [0] --> torre destino [1]`
    * `int count=0;`
      `//conta ciclo para primeira inicialização`
    * `int atualizaStatus=0;`
      `//selecionou origem(0) ou destino(1), ou iniciou jogo`
    * `int range = 0;`
      `//quantos discos torre[i] possui`
    * `float estacaW, estacaH;`
      `//Width e Height das torres`

- **Draw()**:

```
for (int i=0; i<3; i++){
    count++;

    //desenha estacas
    rect(estacaX.get(i), height, estacaW, - (estacaH));

    for (int j=0; j<range; j++){
        if (atualizaStatus==0 && count == 1){
          int d = int(random(sizes.size()));
          float s = base_size + sizes.get(d) * base_size;
          ↪ //algoritmo para gerar tamanhos "aleátorios",
          ↪ 20px(base_size) + random[0,1,2,3,4,5] x 20px
          torres[swap[1]].push(new Disco(j, s, swap[1]));
          ↪ //adiciona novo disco a torre destino
        }
      if (torres[i].empty()==false)
        torres[i].get(j).drawDisco(); //desenha discos
    }
  }
```

- **MouseClicked()**:

```
for (int i=0; i<3; i++){
  if ((mouseX>=estacaX.get(i) && mouseX <=estacaX.get(i)+estacaW) &&
  ↪ (mouseY>=(height-estacaH))){ //Itera cada disco e verifica se
  ↪ mouse está sobre ao clicar
    if (atualizaStatus==1){ //origem já selecionada
      atualizaStatus = 0;
      swap[1] = i;

      if (torres[swap[1]].empty() == false &&
      ↪ torres[swap[0]].empty()==false){
        if (torres[swap[0]].peek().size <=
        ↪ torres[swap[1]].peek().size){
        ↪ //Se ultimo elemento da pilha origem for menor que ultimo
        ↪ da pilha destino
          torres[swap[1]].push(new Disco(torres[swap[1]].size(),
          ↪ torres[swap[0]].peek().size, swap[1]));
          ↪ //adiciona no destino
          torres[swap[0]].pop();
          ↪ //remove da pilha origem
        }
      }else if(torres[swap[0]].empty()==false){
```

```
                torres[swap[1]].push(new Disco(0,
                ↪  torres[swap[0]].peek().size, swap[1]));
                ↪  //adiciona no destino
                torres[swap[0]].pop();
                ↪  //remove da pilha origem
            }
            }else{ //Selecionando origem
               atualizaStatus = 1;
               swap[0] = i;
            }
         }
      }
   }
```

- **Arquivo Disco**: Onde está contida a classe **Disco** responsável pela simbolização de cada disco de forma individual, e sua devida representação visual no jogo.

  - **Disco()**:
    index: posição na estaca (0 - 5)
    size: largura do disco
    estaca: estaca em que o disco se encontra (0,1,2)

```
public class Disco{
    float x, y;
    float size;
    int estaca;

    public Disco(int index, float size, int estaca){
        this.size = size;
        this.estaca = estaca;
        this.x = estacaX.get(estaca)-(size/2);
        this.y = height - (index*0.08*height);
        ↪  //calcula coordenadas com base na estaca/index e tamanho do
        ↪  disco/tela
    }

    void drawDisco(){
        fill(100);
        rect(x, y, size+estacaW, -0.08*width);
        ↪  //Desenha disco considerando largura da estaca
    }
}
```

3 Resultados

- O programa final funciona conforme o esperado, iniciando-se com 5 discos de tamanhos distintos, ao clicar em uma torre e em seguida em outra, caso o disco presente no topo do destino seja maior que o da origem, ou não exista o disco se muda de estaca.

4 Conclusão

- O projeto cumpriu com o esperado e representa bem uma implementação visual do Jogo Torre de Hanói.
- A definição dinâmica de coordenadas e tamanhos dos elementos foi parte mais difícil para mim.
- A implementação foi fundamental para meu entendimento de Stacks e Linked Lists.