Documentação de um Produto de Software

Torre de Hanói

RA-Nome dos Alunos:
822154533 - João Henrique Pacheco de Oliveira
822146121 - Thiago Rodrigues Marinho

2024

ÍNDICE DETALHADO

1. Introdução

1.1. Tema

Desenvolvimento de um algoritmo que utiliza a estrutura de pilha para o projeto de estrutura de dados e análise de algoritmos. Nesse caso foi desenvolvido um algoritmo que simula o brinquedo da Torre de Hanoi.

1.2. Objetivos a serem alcançados

É preciso que o jogo/programa esteja com suas principais funções funcionando corretamente, além disso, é interessante que ofereça um bom desafio para o usuário e que o mesmo tenha recursos visuais apresentando sua interação com o programa e seus resultados/desempenho.

É esperado que o sistema seja desenvolvido na linguagem Java utilizando a IDE NetBeans.

O sistema deve utilizar a estrutura de dados Pilha, nessa pilha é preciso que tenha os métodos push e pop, sendo um deles para empilhar o elemento e o outro para desempilhar (no qual sempre vai desempilhar o elemento que está no topo), além disso é importante ter outros métodos que vão auxiliar no funcionamento, como o método peek(), que retorna elemento que está no topo da pilha, isEmpty() e isFull, que retornam se a pilha está vazia e se a pilha está cheia respectivamente, size() retorna o tamanho máximo de elementos que poderão estar na pilha e por último a função toString() utilizada para apresentar no console como está a pilha.

1.3. Escopo principal

"Torre de Hanói é um quebra-cabeça que consiste em uma base contendo três pinos, em um dos quais são dispostos alguns discos uns sobre os outros, em ordem crescente de diâmetro, de cima para baixo. O problema consiste em passar todos os discos de um pino para outro qualquer, usando um dos pinos como auxiliar, de maneira que um disco maior nunca fique em cima de outro menor em nenhuma situação. O número de discos pode variar sendo que o mais simples contém apenas três." (WIKIPEDIA)

O usuário deve mover os pinos para os três discos. Deve ser implementado a inserção e exclusão de pinos nos discos.

A Torre de Hanói é um jogo estratégico, é preciso pensar antes de cada jogada, talvez até mesmo pensar mais de uma jogada à frente, pois para conseguir passar todos os discos do primeiro pino até o segundo ou terceiro (formando assim uma Torre de Hanói) em um número de tentativas mais baixo é preciso que cada jogada esteja realmente te fazendo progredir no jogo. Por ser um jogo que exige bastante do raciocínio lógico, é comum, principalmente nos iniciantes, o jogador acabar fazendo jogadas redundantes, fazendo com que termine de completar a torre com um número de tentativas/jogadas maior. Em "Torre de Hanói" existem regras importantes e que também serão implementadas em nosso software. Primeiro: Não é possível pegar mais de um disco por vez. Segundo: A Torre de Hanói é formada por discos maiores embaixo dos menores, então não é possível

colocar um disco de diâmetro 5 em cima do disco de diâmetro 2, em nosso jogo cada disco será representado por um número e esse número também representa o tamanho dele. Então no começo o pino 1 estará completo com uma Torre de Hanói e vai ser mostrado ao usuário o seguinte vetor: [5,4,3,2,1], o número 1 é o disco que está no topo, se por acaso eu retirar esse elemento e colocar no pino 2 e depois tentar colocar o disco 2 nesse mesmo pino, então será dito ao usuário que não é possível colocar um disco maior em cima de um menor, ao cometer esse erro o usuário terá que começar o jogo novamente. Essas são as regras da Torre de Hanói, o usuário terá que ter atenção para não descumprir alguma das regras do jogo e jogar estrategicamente para ter o melhor resultado possível.

2. Descrição Funcional

O usuário tem o objetivo de desempilhar os discos que estão no primeiro pino e formar uma torre de Hanói completa em algum dos 2 outros pinos, e claro, sem descumprir a principal regra: discos maiores sempre embaixo dos discos menores, discos menores devem ficar mais ao topo do pino(pilha).

Ao início do jogo o usuário deve escolher em qual dificuldade ele quer jogar, cada dificuldade é identificada por um número e esse mesmo número também representa a quantidade de discos que terá na Torre de Hanói, ou seja, se eu escolho a dificuldade 3 - Molezinha (digitando o número 3 e clicando em "ok") o jogo inicia o pino 1 com os valores, [3,2,1], então nessa dificuldade o jogo se torna bem mais fácil pois é preciso empilhar apenas 3 discos. O jogo conta com as seguintes dificuldades disponíveis: Molezinha - 3 discos, Interessante - 4 discos, Desafiador - 5 discos, SENHOR RACIOCÍNIO LÓGICO - 6 discos.

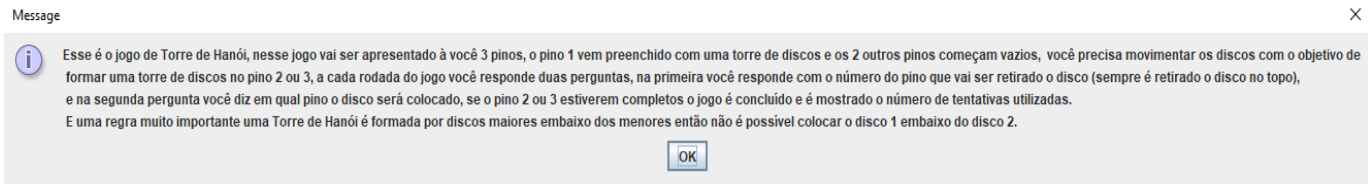
Após escolher a dificuldade, o programa cria as pilhas de acordo com a dificuldade escolhida, um adendo, cada pino é um objeto criado da classe Pilha que terá todos os métodos e atributos para o funcionamento correto do sistema. Após a criação dos objetos pino 1, pino 2 e pino 3, o sistema realiza uma instrução para o usuário através da de uma JOptionPane de Input, a instrução é "Escolha de qual pino você quer retirar um disco", nessa parte o usuário deve digitar em uma caixa de mensagem o número do pino que será retirar o disco mais ao topo (ao lado terá informações de cada pino para o usuário não se perder), logo em seguida, o programa realiza outra instrução que é "Digite o número do pino que você quer colocar esse disco", nessa parte é necessário o usuário informar qual pino terá o disco que foi retirado do topo de outro pino.

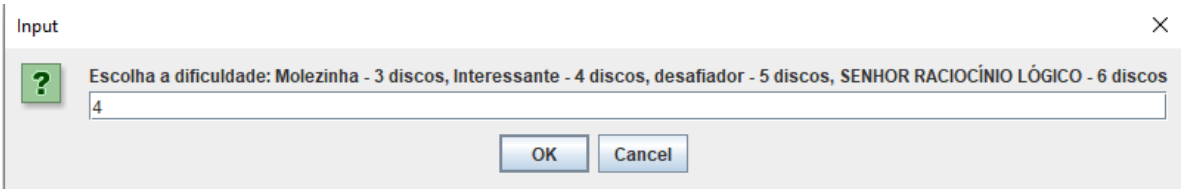
Ao final desses dois comandos, o sistema informa ao usuário com caixas de mensagem como está ficando cada pino e também seu número de tentativas. Depois disso o sistema volta a realizar as duas instruções explicadas no parágrafo anterior, ou seja, o usuário ficará nesse loop várias vezes, loop acaba quando o jogo acaba e o jogo acaba quando o usuário consegue deixar o pino 2 ou o pino 3 cheios, ao concluir com sucesso o jogo da Torre de Hanói é apresentado através de uma JOptionPane de mensagem a seguinte mensagem "MUITO BEM! Você empilhou a Torre de Hanói em um total de (número de tentativas) tentativas". Observação: O jogo também finaliza se você cometer o erro de tentar empilhar um disco maior em cima de um disco menor.

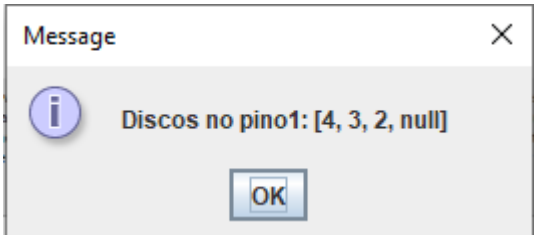
2.1 Requisitos não funcional

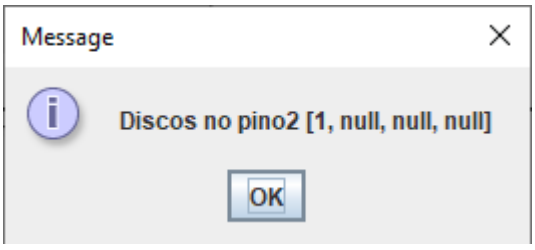
O software deve ser desenvolvido na linguagem Java e na IDE do NetBeans. Sistema Operacional foco da aplicação é o Windows.

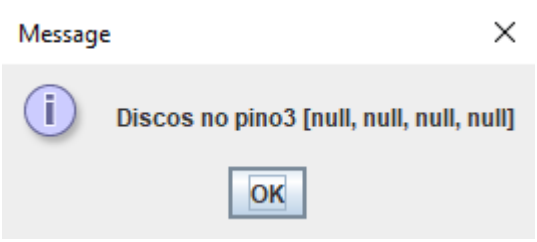
3. Interfaces

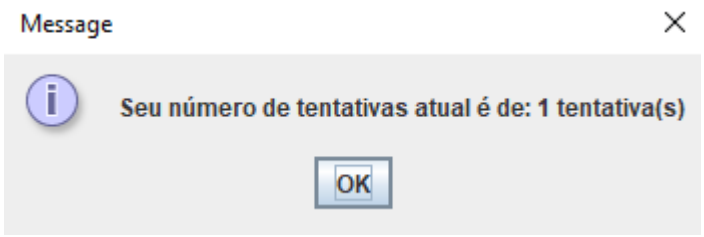


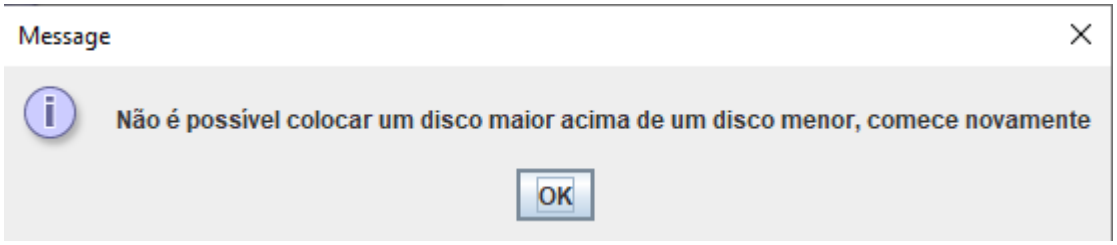


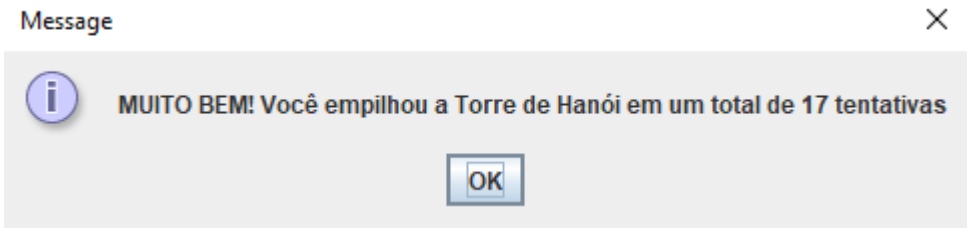







Message
Seu número de tentativas atual é de: 1 tentativa(s)
OK


Message
Não é possível colocar um disco maior acima de um disco menor, comece novamente
OK


Message
MUITO BEM! Você empilhou a Torre de Hanói em um total de 17 tentativas
OK

4. Anexo

https://drive.google.com/drive/folders/1GBAQkuCiy084YiINUTiS9R1QtTuMtVEq?usp=sharing

```
package torrehanoia3;

import javax.swing.JOptionPane;


public class TorreHanoiA3 {



   public static void main(String[] args) {


     JOptionPane.showMessageDialog(null, "Esse é o jogo de Torre de Hanói, nesse jogo vai ser apresentado à você 3 pinos, o pino 1 vem preenchido com uma torre de discos e os 2 outros pinos começam vazios,  você precisa movimentar os discos com o objetivo de\n formar uma torre de discos no pino 2 ou 3, a cada rodada do jogo você responde duas perguntas, na primeira você responde com o número do pino que vai ser retirado o disco (sempre é retirado o disco no topo),\n e na segunda pergunta você diz em qual pino o disco será colocado, se o pino 2 ou 3 estiverem completos o jogo é concluído e é mostrado o número de tentativas utilizadas.\n E uma regra muito importante uma Torre de Hanói é formada por discos maiores embaixo dos menores então não é possível colocar o disco 1 embaixo do disco 2.");
       int numeroDiscos = Integer.parseInt(JOptionPane.showInputDialog(null, "Escolha a dificuldade: Molezinha - 3 discos, Interessante - 4 discos, desafiador - 5 discos, SENHOR RACIOCÍNIO LÓGICO - 6 discos"));
       Pilha pino1 = new Pilha(numeroDiscos);
       Pilha pino2 = new Pilha(numeroDiscos);
       Pilha pino3 = new Pilha(numeroDiscos);
       if(numeroDiscos == 3){
          pino1.push(3);
          pino1.push(2);
          pino1.push(1);
       }
       else if(numeroDiscos == 4){
          pino1.push(4);
          pino1.push(3);
          pino1.push(2);
          pino1.push(1);
       }
       else if(numeroDiscos == 5){
          pino1.push(5);
          pino1.push(4);
          pino1.push(3);
          pino1.push(2);
          pino1.push(1);
       }
       else if(numeroDiscos == 6){
          pino1.push(6);
          pino1.push(5);
          pino1.push(4);
```

```
            pino1.push(3);
            pino1.push(2);
            pino1.push(1);
        }
        int tentativas = 0;
        int  aux = 0;
        while(!pino2.isFull() && !pino3.isFull() && aux==0){
        int topoHanoi = 0;
        int escolhaRetirada = Integer.parseInt(JOptionPane.showInputDialog("Escolha de qual pino você quer
retirar um disco -"+ pino1.toString(" pino 1: ")+pino2.toString(" pino 2: ") + pino3.toString(" pino 3: ")));
            if(escolhaRetirada == 1){
               topoHanoi = pino1.peek();
               pino1.pop();
            }
            else if(escolhaRetirada == 2){
               topoHanoi = pino2.peek();
               pino2.pop();
            }
            else if(escolhaRetirada == 3){
               topoHanoi = pino3.peek();
               pino3.pop();
            }
        int escolhaDestino = Integer.parseInt(JOptionPane.showInputDialog("Digite o número do pino que você
quer colocar esse disco -"+ pino1.toString(" pino 1: ")+pino2.toString(" pino 2: ") + pino3.toString(" pino 3:
")));
               if(escolhaDestino == 1){
                  aux = pino1.push(topoHanoi);
               }
               else if(escolhaDestino == 2){
                  aux = pino2.push(topoHanoi);
               }
               else if(escolhaDestino == 3){
                  aux = pino3.push(topoHanoi);
               }


            tentativas++;
            JOptionPane.showMessageDialog(null, "Discos no "+pino1.toString("pino1: "));
            JOptionPane.showMessageDialog(null, "Discos no "+pino2.toString("pino2 "));
            JOptionPane.showMessageDialog(null, "Discos no "+pino3.toString("pino3 "));
            JOptionPane.showMessageDialog(null, "Seu número de tentativas atual é de: "+tentativas+ "
tentativa(s)");
            System.out.println(pino1.toString("pino 1:"));
            System.out.println(pino2.toString("pino 2:"));
            System.out.println(pino3.toString("pino 3:"));
            System.out.println("número de tentativas: "+tentativas);
            if(pino2.isFull() || pino3.isFull()){
               JOptionPane.showMessageDialog(null, "MUITO BEM! Você empilhou a Torre de Hanói em um
total de "+tentativas+" tentativas");
               System.out.println("MUITO BEM! Você empilhou a Torre de Hanói em um o total de
"+tentativas+" tentativas");
            }
        }
        }
```

```
    }



package torrehanoia3;

import java.util.Arrays;
import javax.swing.JOptionPane;


public class Pilha {


    private int maxSize;
    private Integer[] torre;
    private int top;


    public Pilha(int tam){
        maxSize = tam;
        torre = new Integer[maxSize];
        top = -1;
    }

    public int push(int disco){
        if(isFull()){
            System.out.println("Stack cheia");
            return 0;
        }
        else if(!isEmpty() && peek() <= disco){
          JOptionPane.showMessageDialog(null, "Não é possível colocar um disco maior acima de um disco
menor, comece novamente");
          return -1;
        }
        if(isEmpty()){
            top++;
            torre[top]=disco;

        }
        else if(!isEmpty() && peek() >= disco){
            top++;
            torre[top]=disco;
        }

        return 0;
    }
   public Integer pop(){
       if(isEmpty()){
          JOptionPane.showMessageDialog(null, "Não é possível retirar discos de um pino sem discos");
          return -1;
       }
       int oldTop = top;
       top--;
```

```
        torre[oldTop]= null;
        return torre[oldTop];
        }
    public Integer peek(){
        if(isEmpty()){
            return -1;
        }
        return torre[top];
    }

    public boolean isEmpty(){
        return(top == -1);
    }

    public boolean isFull(){
        return(top == maxSize - 1);

    }
    public int size(){
        return maxSize;
    }

    public String toString(String nomePino){
        return nomePino + Arrays.toString(torre);
    }

}
```

Referência Bibliográfica

Java Swing - JOptionPane

Javax.Swing.JOptionPane – Conhecendo e utilizando a classe JOptionPane | Bruno Augusto.