**PONTIFÍCIA UNIVERSIDADE CATÓLICA DE CAMPINAS**

**BACHARELADO EM SISTEMAS DA INFORMAÇÃO**

**DIEGO DA SILVA BARBOSA**

**JÚLIO RENATO MINGUINI FAÚNDES**

**VINICIUS NATALINO ZACHEU**

**FILA ORDENADA DE ATENDIMENTO TELEFÔNICO EM JAVA**

**CAMPINAS**

**2021**

**Introdução.**

Este relatório busca descrever de maneira prática, clara e direta como uma fila ordenada de atendimento telefônico foi construída utilizando a linguagem Java. A fila é ordenada dando prioridade para chamadas de origens mais distantes (com base no DDD) e, em seguida, por ordem de chegada.

**Considerações Importantes:**

Este programa é capaz receber apenas números nacionais. Além disso é considerado que o DDD possui apenas 2 dígitos.

**Regras do jogo.**

O jogo da velha deve ser jogado sempre por dois jogadores. Neste caso em específico, o game será jogado entre um jogador humano e um programa ou IA (Inteligência Artificial) e cada um joga um símbolo diferente, comumente representados por um ‘X’ e um ‘O’. O tabuleiro consta com nove lacunas que serão preenchidas de maneira intercalada pelos jogadores e o jogador que inicia a partida é escolhido de maneira aleatória.

O objetivo do jogo é formar uma linha constituída de três símbolos iguais. Esta linha pode ser na horizontal, na vertical e nas diagonais. Visto que a condição de vitória é a formação destas linhas, é importante que os jogadores busquem alcançar a vitória ao mesmo tempo em que bloqueiam linhas do adversário. Existe, também, a possibilidade de empate. O empate acontece quando nenhum dos dois jogadores conseguiram formar linhas durante a partida.

**O programa.**

Assim que o jogo é iniciado, o usuário é apresentado com a tela principal do jogo – apelidada de “Lobby do Jogo da Velha”. Nesse lobby, o jogador escolhe se irá jogar ou sair do jogo e, após a primeira partida, também será apresentado um placar que mostra o número de vitórias, de derrotas, de empates e de partidas. Quando o jogador seleciona a opção jogar, o programa executa uma função chamada jogo – a qual retorna três possíveis valores: ‘0’ para empates, ‘1’ para derrotas e ‘2’ para vitórias.

Quando a função jogo é executada, ela inicia uma matriz chamada tabuleiro e logo em seguida executa uma função que *reseta* o tabuleiro, ou seja, assegura que todos os valores presentes na matriz são iguais a ‘0’. Depois disso o programa define de maneira aleatória qual jogador irá iniciar a partida primeiro.

O programa, então, inicia um loop em for que será executado, no máximo, 9 vezes – número máximo de jogadas possíveis antes de determinar o empate. Se for a vez do jogador ele irá jogar e, então, o programa checará se sua jogada resultou numa vitória. Se não, o jogo troca a vez para a IA e repete o mesmo processo até que o número de jogadas atinja 9 ou algum dos dois vença antes disso. A depender do resultado, o programa retorna ‘0’, ‘1’ ou ‘2’ como explicado anteriormente.

**A inteligência artificial.**

A inteligência artificial do programa trabalha através de prioridades. Ela primeiro verifica se existe alguma forma dela vencer, ou seja, se existem 2 símbolos iguais ao dela alinhados em alguma possível linha e que haja uma lacuna que ela possa preencher nessa rodada. Caso não seja possível vencer nessa rodada, ela verifica se é possível bloquear o adversário, ou seja, ela verifica as mesmas condições que anteriormente, mas olhando especificamente para os símbolos do jogador a fim de impedir sua vitória. Por fim, caso não seja possível nem vencer, nem bloquear, o computador joga aleatoriamente em uma lacuna vazia – isso foi implementado dessa forma a fim de dar a impressão ao jogador de que os jogos são mais orgânicos e nem sempre acontecem da mesma forma.

**O jogador.**

A vez do jogador é simples. Ele define as coordenadas de sua jogada e, a partir disso, seu símbolo é posicionado no tabuleiro.

**Vitória ou derrota.**

Para definir se o jogo resultou em vitória ou derrota para o jogador, o programa trabalha através de somas de linhas, colunas e diagonais. Sempre que o computador joga, ele adiciona o valor ‘1’ na posição da matriz em que fez sua jogada. Da mesma forma, sempre que o jogador joga, o programa adiciona o valor ‘4’ na posição. Esses números foram escolhidos dessa forma para garantir que não existem possibilidades da soma de uma linha resultar em ‘3’ ou ‘12’ caso os três elementos verificados não sejam iguais. Assim, quando a soma de uma linha resulta em ‘12’, é vitória do jogador e, quando resulta em ‘3’, é derrota do jogador.

**Referências.**

O programa foi construído inteiramente a partir das noções do jogo do autor.

**O programa.**

|  |
| --- |
| public class FilaDeChamadas {  Nodo prim = null;  int tamanho = 0;  public void inserir(String telefone) throws Exception {  Nodo novo = new Nodo();  Nodo ant = null;  for (Nodo i = prim; i != null && getPrioridade(i.getInfo()) < getPrioridade(telefone); i = i.getProx()) {  ant = i;  }  novo.setInfo(telefone);  if (ant == null) {  novo.setProx(prim);  prim = novo;  } else {  novo.setProx(ant.getProx());  ant.setProx(novo);  }  tamanho++;  }  public String remover() {  String telefone = prim.getInfo();  prim = prim.getProx();  tamanho--;  return telefone;  }  public boolean validacaoTelefone(String telefone) {  char carac;  for (int i = 0; i < telefone.length(); i++) {  carac = telefone.charAt(i);  if (!Character.isDigit(carac)) {  return false;  }  }  return true;  }  public String getPosicao(String telefone) {  int posicao = 1;  for (Nodo i = prim; i != null && !i.getInfo().equals(telefone); i = i.getProx()) {  posicao++;  }  return posicao + "/" + this.tamanho;  }  public boolean inFila(String telefone) {  for (Nodo i = prim; i != null; i = i.getProx()) {  if (i.getInfo().equals(telefone))  return true;  }  return false;  }  public boolean isVazia() {  if (prim == null)  return true;  return false;  }  // Quanto mais longe a chamada, menor o valor (Quanto menor, mais prioridade);  public int getPrioridade(String telefone) throws Exception {  switch (getDDD(telefone)) {  case 95:  return 1;  case 96:  return 2;  case 68:  return 3;  case 97:  return 4;  case 92:  return 5;  case 91:  return 6;  case 98:  return 7;  case 85:  return 8;  case 84:  return 9;  case 83:  return 10;  case 81:  return 11;  case 88:  return 12;  case 86:  return 13;  case 69:  return 14;  case 93:  return 15;  case 94:  return 16;  case 99:  return 17;  case 82:  return 18;  case 87:  return 19;  case 89:  return 20;  case 79:  return 21;  case 71:  return 22;  case 75:  return 23;  case 74:  return 24;  case 63:  return 25;  case 65:  return 26;  case 66:  return 27;  case 62:  return 28;  case 77:  return 29;  case 73:  return 30;  case 53:  return 31;  case 55:  return 32;  case 51:  return 33;  case 54:  return 34;  case 61:  return 35;  case 64:  return 36;  case 67:  return 37;  case 33:  return 38;  case 38:  return 39;  case 49:  return 40;  case 46:  return 41;  case 48:  return 42;  case 47:  return 43;  case 42:  return 44;  case 45:  return 45;  case 27:  return 46;  case 28:  return 47;  case 41:  return 48;  case 43:  return 49;  case 44:  return 50;  case 34:  return 51;  case 37:  return 52;  case 31:  return 53;  case 32:  return 54;  case 22:  return 55;  case 21:  return 56;  case 24:  return 57;  case 17:  return 58;  case 18:  return 59;  case 13:  return 60;  case 12:  return 61;  case 35:  return 62;  case 16:  return 63;  case 14:  return 64;  case 15:  return 65;  case 11:  return 66;  case 19:  return 67;  default:  throw new Exception("DDD inválido.");  }  }  private int getDDD(String telefone) {  return Integer.parseInt(telefone.substring(0, 2));  }  @Override  public String toString() {  if (isVazia())  return "{}";  return "{" + this.prim.getInfo() + "}";  }  private class Nodo {  private String info;  private Nodo prox;  public String getInfo() {  return info;  }  public void setInfo(String info) {  this.info = info;  }  public Nodo getProx() {  return prox;  }  public void setProx(Nodo prox) {  this.prox = prox;  }  }  } |

**Exemplo de programa que utiliza a classe desenvolvida.**

|  |
| --- |
| import java.util.Scanner;  import java.util.concurrent.TimeUnit;  public class App {  public static void main(String[] args) {  String telefone = "";  FilaDeChamadas fila = new FilaDeChamadas();  try {  Scanner ler = new Scanner(System.in);  boolean loop = true;  clear();  do {  System.out.println("Seja bem-vindo ao atendimento ao cliente! :)\n");  System.out.println("Opções:");  System.out.println("[0] - Sair");  System.out.println("[1] - Inserir novo número");  switch (Integer.parseInt(ler.nextLine())) {  case 0:  System.out.println("Saindo do programa. . .");  System.exit(1);  case 1:  clear();  System.out.printf("Telefone: ");  telefone = ler.nextLine();  while (!fila.validacaoTelefone(telefone)) {  clear();  System.out.println("Por favor, digite um número válido :)\n");  System.out.printf("Telefone: ");  telefone = ler.nextLine();  }  loop = false;  default:  break;  }  if (loop) {  clear();  System.out.println("Digite uma opção válida. . .");  }  } while (loop);  // Insere nove números + o digitado  fila.inserir("19997713109");  fila.inserir("99914568771");  fila.inserir("95998654758");  fila.inserir("35996853214");  fila.inserir("11998547123");  fila.inserir("24998652347");  fila.inserir("47997456245");  fila.inserir("19996325477");  fila.inserir("99996325787");  fila.inserir(telefone);  clear();  while (!fila.isVazia()) {  System.out.println("----------------------------------------------");  if (fila.inFila(telefone)) {  if (fila.getPosicao(telefone).charAt(0) == '1') {  System.out.println("O seu número está sendo atendido!!");  TimeUnit.SECONDS.sleep(2);  fila.remover();  continue;  } else {  System.out.println(  "Seu número {" + telefone + "} está na posição " + fila.getPosicao(telefone) + "\n");  }  }  System.out.println("Número sendo atendido: " + fila);  TimeUnit.SECONDS.sleep(2);  fila.remover();  }  System.out.println("Todos os números já foram atendidos.");  ler.close();  } catch (Exception e) {  System.err.println(e.getMessage());  }  }  static void clear() {  for (int i = 0; i < 100; i++) {  System.out.println();  }  }  } |