|  |
| --- |
| **Parallel Distributed Computing**  **Second Review**  **Rapid Roll Game using multithreading**  **970px-Vellore_Institute_of_Technology_seal.svg.png**  **Team member –**  **Shubham Goel 15BCE0296**  **Satyam Chouksey 15BCE0838** |
|  |
| **Abstract**  A parallel programming model is an abstraction of parallel computer architecture, with which it is convenient to express algorithms and their composition in programs. The value of a programming model can be judged on its generality that how well a range of different problems can be expressed for a variety of different architectures, and its performance. Also it determines that how efficiently the compiled programs can execute. The implementation of a parallel programming model can take the form of a library invoked from a sequential language, as an extension to an existing language, or as an entirely new language.  This game is developed in JAVA and it uses MULTITHREADING and AWT. Multithreading involve technique by which a single set of code can be used by several processors at different stages of execution. And AWT (Abstract Window Toolkit) is an API to develop GUI or window based application in JAVA.  **Sypnosis:**  **Modules involved :**  Java Multithreading is mostly used in games, animation etc. Rapid roll game is a platform game, the aim of which is to operate a ball, move on the blocks, overcome the obstacle and to not let it fall down from the platform. The ball is operated using left and right cursor in the keyboard.  **java.awt.event.** (to run the thread with more speed)  **thread t** (to create a new thread in java)  **t = new Thread(this,"game");**  Type of parallelism used : We will simultaneously execute two or more parts of a program for maximum utilization of the CPU time. A multithreaded program contains two or more parts that can run concurrently. Each such part of a program called thread. In Multithreaded environment, programs that are benefited from multithreading, utilize the maximum CPU time so that the idle time can be kept to minimum.  **Key Points for the type of parallelization:**  The thread-level or task-level parallelism is used in our project.   * Multiple threads for the same application are executed concurrently.   **Code:**  import java.awt.\*; //CHANGE THE scrollSpeed value  import java.awt.event.\*; //to run the thread with more speed  import java.util.\*; **import all the classes in the util packages**  import java.io.\*;**import the classes defned in java.io package to your file**  class HighSc  {  int hsc;  BufferedReader br; **//simplest way to read a file**  FileInputStream fis; **// create new input file stream**  String sths;  public String getHighScore() throws IOException  {  fis = new FileInputStream("highscore.txt");  br = new BufferedReader(new InputStreamReader(fis));  sths = br.readLine(); //**read text from a character based input stream**  fis.close();//**file input strean**  return sths;  }  }  class GameFrame extends Frame implements Runnable, KeyListener  ,ActionListener  {  MenuBar mb;  Menu m1;  MenuItem mi1,mi2,mi3,mi4,mi5,mi6;  Button b1,b2,b3;  Button ball;  Button oneUp;  final int TOPLINE = 50; //**fixing of the screen of the game run**  final int BOTTOMLINE = 350; //**fixing of the screen of the game run**  final int LEFTLINE = 20; //**fixing of the screen of the game run**  final int RIGHTLINE = 350 ; //**fixing of the screen of the game run**  int scrollSpeed = 25;  int x=50, y=300;  int x2=125,y2=200;  int x3 = 90,y3=100;  int bx = 80;  int by = 239;  int ox = 0;  int oy = 0;  int score = 0;  int oneUpCount = 1;  int t1,t2,t3;  int chances=3;  int a,b,c;  int diff = 65;  String msg = "";  String chns = "";  String st = "";  char ch;  int kcode;  boolean flagLKey = true; **//turn on key left**  boolean flagRKey = true; //**turn on key right**  boolean flagTop1 = false; // **turn off for other keyboard keys**  boolean flagTop2 = false;  boolean flagTop3 = false;  boolean flagDrop = true;  boolean flagMove = true;  boolean flagBetween = true;  boolean flagOnx = true;  boolean flagOnx2 = false;  boolean flagOnx3 = false; **// on the axis**  boolean flagJump = true;  boolean flagNew1 = false;  boolean flagOneUp = false;  Thread t;  GameFrame()  {  mb = new MenuBar();  m1 = new Menu("File"); //**creates menu**  mi1 = new MenuItem("New Game"); //**defining objects in menu**  mi2 = new MenuItem("HighScores"); //**defining objects in menu**  mi3 = new MenuItem("Exit"); //**defining objects in menu**  setMenuBar(mb);  mb.add(m1);  m1.add(mi1);  m1.add(mi2);  m1.add(mi3);  setTitle("PDC Project");  setSize(400,350);  setLayout(null);  b1 = new Button("");  b2 = new Button(""); **//creates a button**  b3 = new Button(""); **//creates a button**  ball = new Button("o"); **//creates a button**  oneUp = new Button("0"); **//creates a button**  add(ball);  add(b1);  add(b2);  add(b3);  add(oneUp);  b1.setBounds(x,y,70,20); **//fixing of the movement of bars**  b2.setBounds(x2,y2,70,20);  b3.setBounds(x3,y3,70,20);  ball.setBounds(bx,by,10,10);  oneUp.setBounds(bx-5,by,5,5);  setBackground(Color.blue); //**setting color of the background**  setForeground(Color.white); **//setting color of the bar**  ball.addKeyListener(this); //**to change the state of key**  mi1.addActionListener(this); **//to notify when you click on button**  mi2.addActionListener(this); **//to notify when you click on button**  mi3.addActionListener(this); **//to notify when you click on button**  addWindowListener(new WindowAdapter()  {  public void windowClosing(WindowEvent we)  {  closeAll();  }  });  t = new Thread(this,"game"); //**creates a new thread**  t.start();  } /**/end of constructor**  public void actionPerformed(ActionEvent ae)  {  st = ae.getActionCommand();  if(st.equals("New Game")) //**sets to new game when clicked**  {  stop();this.setVisible(false);  Frame ng = new GameFrame(); **// predefined function for game**  ng.setVisible(true);  }  else if(st.equals("HighScores"))  {  HighSc hs = new HighSc();  try  {  msg = "";  msg = hs.getHighScore(); repaint(); **//msg to give high score**  }  **Journal reference**   * J. D. R. III and P. Haggar. Multithreaded exception handling in java. Java Report, Aug. 1998. * D. Lea. Concurrent Programming in JavaTM, Second Edition: Design Principles and Patterns. Addison-Wesley, 1999. * K. Yi and B.-M. Chang. Exception analysis for java. In A. Moreira and D. Demeyer, editors, Object-Oriented Technology. ECOOP’99 Workshop Reader (Formal Techniques for Java Programs), volume 1743 of Lecture Notes in Computer Science, pages 111–112. Springer-Verlag, June 1999. * J. Zalewski IEEE Parallel & Distributed Technology: Systems & Applications, 1996 * C. Boyapati and M. Rinard. A parameterized type system for race-free Java programs. In Proceedings of the 16th Annual Conference on Object-Oriented Programming Systems, Languages and Applications, Tampa Bay, FL, Oct. 2001. |
|  |
|  |
|  |

|  |
| --- |
|  |