DOCUMENTATION

TRON GAME

ARBAB ALI

D1CPLD

Description:

Create agame, with we can play the light-motorcycle battle(known from the Tron movie) in a top view.Two players play against each other with two motors, where each motor leavesa light trace behind of itself on the display.The motor goes in each seconds toward the direction, that the player has set recently. The first player can use the WASD keyboard buttons, while the second one can use the cursor buttons for steering.A player loses if its motor goes to the boundary of the game level, or it goes to the light trace of the other player.Ask the name of the players before the game starts, and let them choose the color their light traces. Increase the counter of the winner by one in the database at the end of the game. If the player does not exist in the database yet, then insert a record for him. Create a menu item, which displays a highscore table of the players for the 10 best scores.Also, create a menu item which restarts the game.

User Doc:

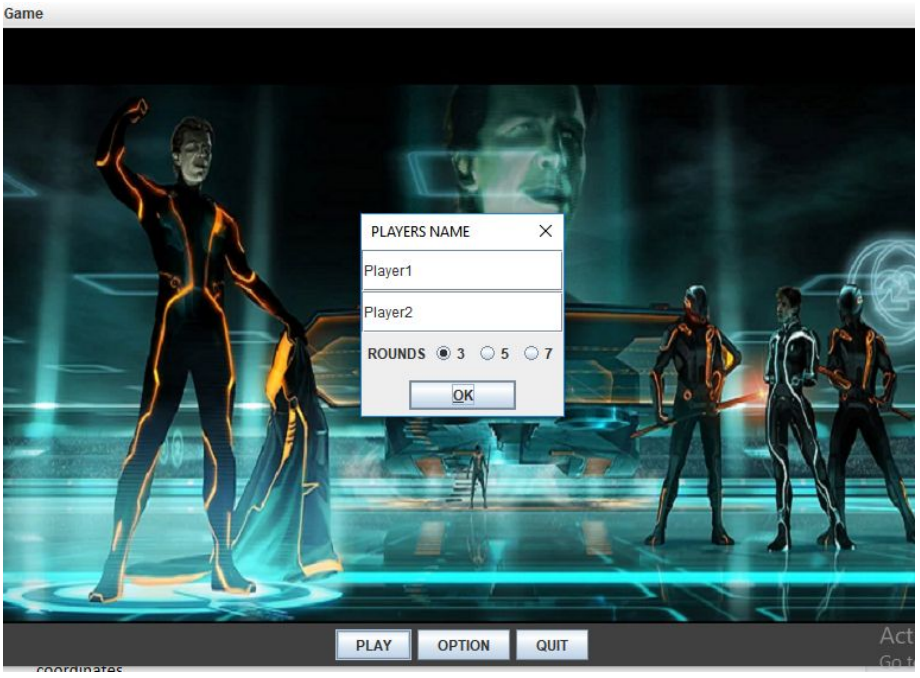
To play this game, you need to use keyboard keys: Up, Left, Down, Right for Player1 and W,A,S,D for Player2.
Players can make a score by making another bike collide with walls or LightTrace . Player score increases by one each game . If you collide with boundaries , or another motorcycle or its trace or itself track , then your round is over and that player dies.

Your program should be user friendly and easy to use. You have to make an object oriented implementation,

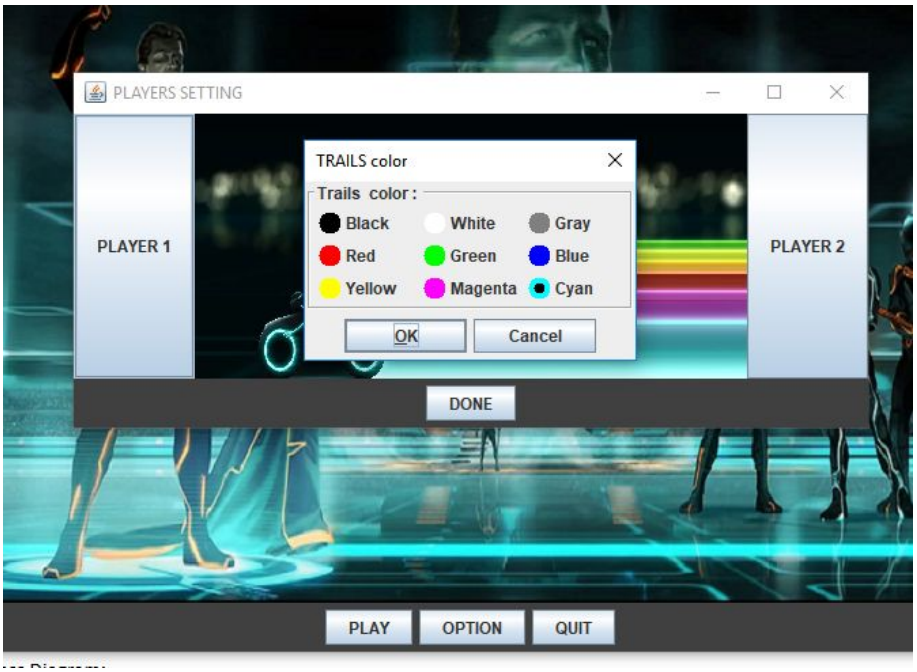
There is a mainMenu you can play game by clicking on the Play button Before Game starts there is a dialog menu to ask for player names and rounds to play.

Ask the name of the players before the game starts, and let them choose the color of their light traces.

You can select any of these radio buttons for each round default is 3



THE COLOR OF PLAYERS TRACE CAN BE SELECTED BY CLICKING ON OPTIONS BUTTON



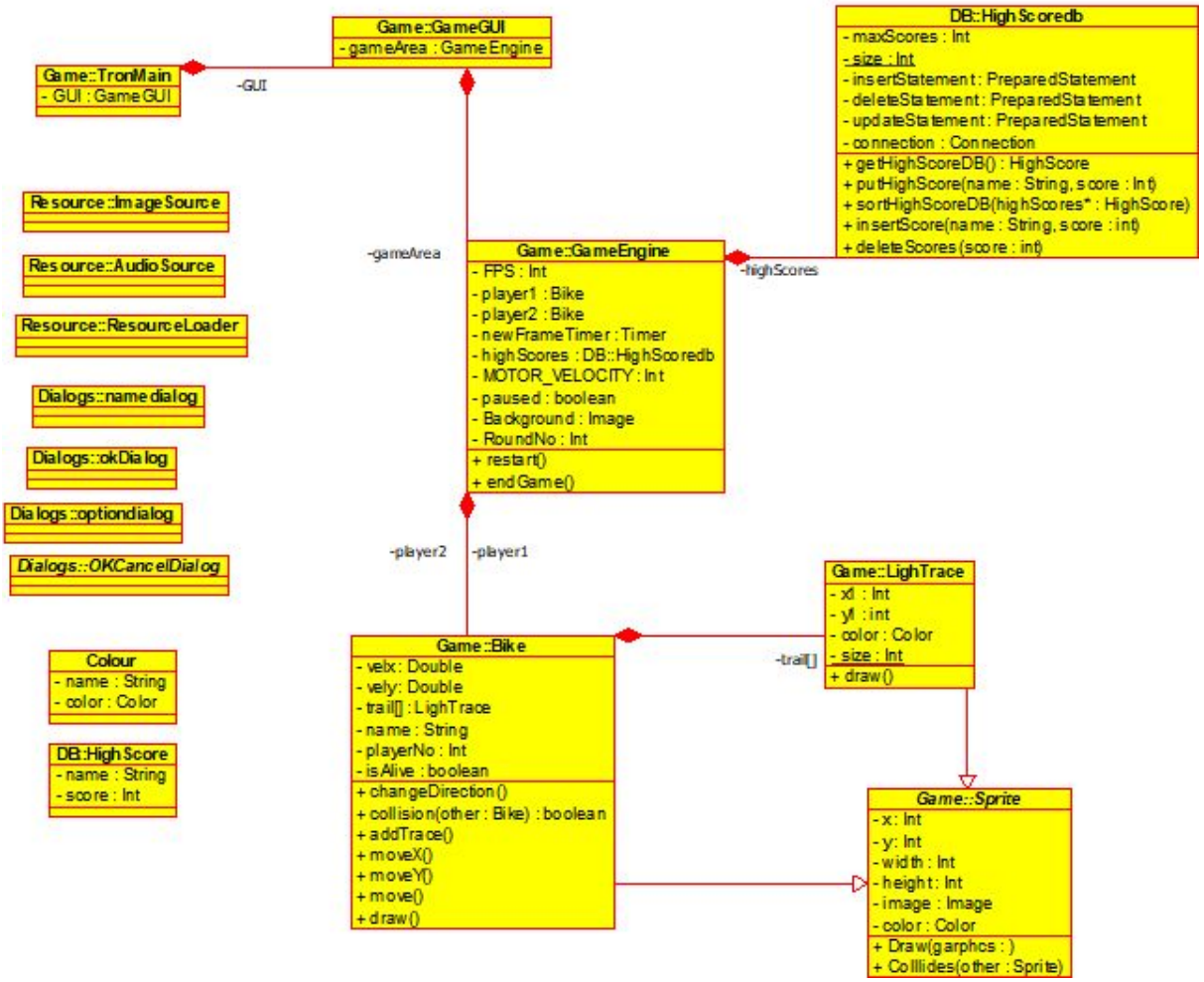
The Game Board is 2D retro graphic and the LightTraces colors are chosable from the options menu.

Class Diagram:

There is a base class sprite which saves the basic coordination position and also draws the image on board and checks if it intersects with another sprite class child.

The LighTrace class extends Sprite class and overrides the draw method to draw a rectangular shape with constant height and width.The Bike class extends the Sprite class and overrides the draw method to draw the motor sprite image as well as the lightTrace line behind visited coordinates.

The GameEngine is the main game Board functionality where the player1 and player2 instance are created with specified round No as well FPS .The Game GUI is main Menu and has most of functionality regard the Graphics and menus HighScoreDB is class created to use Database derby to store the high score value with name in a table

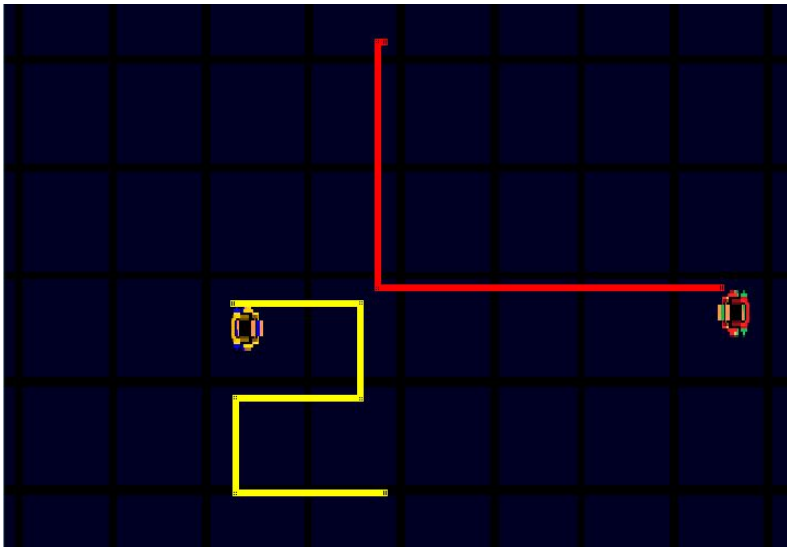

Game::GameGUI
- gameArea : GameEngine
Game::TronMain
- GUI : GameGUI
-GUI
DB::HighScoredb
- maxScores : Int
- size : Int
- insertStatement : PreparedStatement
- deleteStatement : PreparedStatement
- updateStatement : PreparedStatement
- connection : Connection
+ getHighScoreDB() : HighScore
+ putHighScore(name : String, score : Int)
+ sortHighScoreDB(highScores* : HighScore)
+ insertScore(name : String, score : int)
+ deleteScores(score : int)
Resource::ImageSource
Resource::AudioSource
Resource::ResourceLoader
Dialogs::namedialog
Dialogs::okDialog
Dialogs::optiondialog
Dialogs::OKCancelDialog
-gameArea
Game::GameEngine
- FPS : Int
- player1 : Bike
- player2 : Bike
- newFrameTimer : Timer
- highScores : DB::HighScoredb
- MOTOR_VELOCITY : Int
- paused : boolean
- Background : Image
- RoundNo : Int
+ restart()
+ endGame()
-highScores
-player2
-player1
Game::LighTrace
- xl : Int
- yl : int
- color : Color
- size : Int
+ draw()
-trail[]
Game::Bike
- velx: Double
- vely: Double
- trail[] : LighTrace
- name : String
- playerNo : Int
- isAlive : boolean
+ changeDirection()
+ collision(other : Bike) : boolean
+ addTrace()
+ moveX()
+ moveY()
+ move()
+ draw()
Colour
- name : String
- color : Color
DB::HighScore
- name : String
- score : Int
Game::Sprite
- x: Int
- y: Int
- width : Int
- height : Int
- image : Image
- color : Color
+ Draw(garphcs :)
+ Colllides(other : Sprite)

You have to use simple graphics for the game display. The "sprite"of the player's character should be able to move with the well known WASD keyboard buttons.You can also implement mouse event handlers to other functions of the game.

SCREENSHOTS OF GAME:

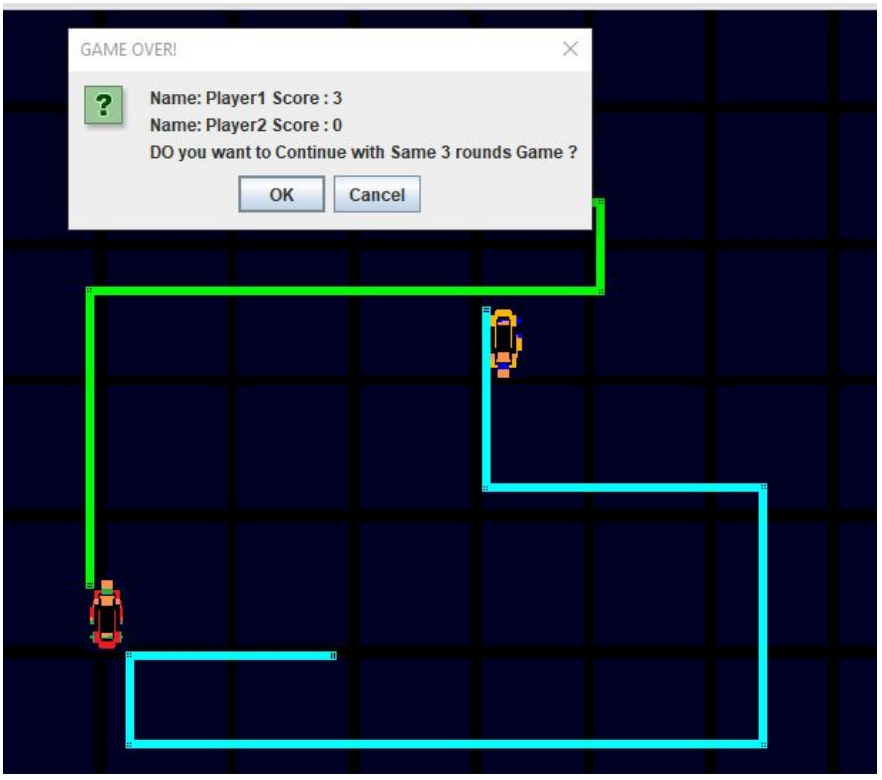
Board graphics are used like Retro TRON game The player 1 can be moved with arrow keys

Player 2 can be moved with WASD keys.



Each game needs to have a timer, which counts the elapsed time since the start of the game level.

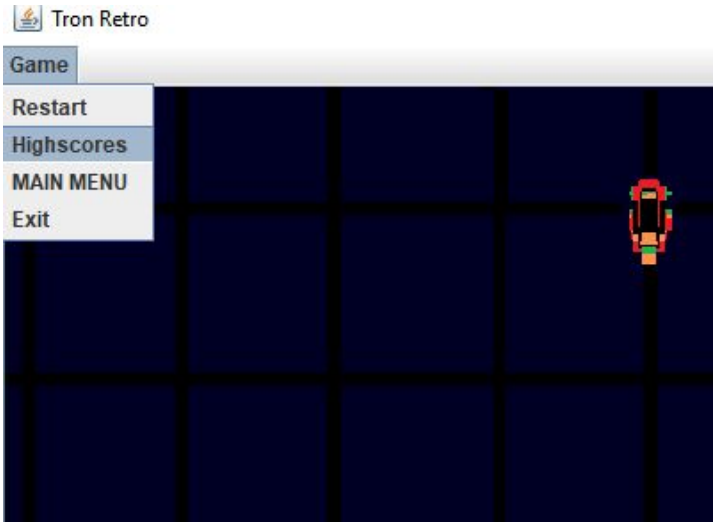
There is timer that creates new frame or repaints the frame every second or mili



Increase the count of the winner by one in the database at the end of the game. If the player does not exist in the database yet, then insert a record for him. Create a menu item, which displays a highscore table of the players for the 10 best scores.Also, create a menu item which restarts the game

There is menu bar in game which has button to restart current game with initial values and also a menu for high score table

For highscore storage i used Derby client where i store the value and access and updates them with HighScoreDB class shown in UML



by clicking on high score a window of top 10 scores in ascending sorting order appears

Implementation:

HighScoreTable Menu:

Creates a new menu item frame window and prints the values of table retrieved from getHighScoresDB

```
            highscoreMenuItem.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent ae) {
                JFrame newFrame= new JFrame();
                newFrame.setTitle("HIGH SCORE TABLE !");
                JLabel contentPane = new JLabel();
                contentPane.setIcon( ImageSource.getIcon("res/Data/highscore2.png") );
                contentPane.setLayout( new GridLayout(6,2) );
                JLabel header= new JLabel("10!           ");
                JLabel empty=new JLabel("                        TOP");
                header.setFont(new Font("Dialog", Font.BOLD, 25));
                empty.setFont(new Font("Dialog", Font.BOLD, 25));
                empty.setForeground(Color.red);
                header.setForeground(Color.red);
                contentPane.add(empty);
                contentPane.add(header);
        DefaultListModel<String> l1 = new DefaultListModel<>();
        AudioSource.playClip("src/res/data/Scores.wav");
                try{
                HighScoreDB highScores=new HighScoreDB( 10);
                highScores.getHighScoreDB();

                        for(Integer i=1;i<=highScores.size;i++)
                                {
                                        if(i>10)
                                        break;
                                JLabel elem=new JLabel( i.toString()+ "->"+
highScores.getHighScoreDB().get(i-1).toString());
                                //elem.setBackground(Color.yellow);
                                elem.setFont(new Font("Dialog", Font.BOLD, 20));
                                elem.setForeground(Color.YELLOW);
                                contentPane.add(elem );
                                //System.out.println(highScores.getHighScores().get(i).toString());
                                }
                }
                        catch (SQLException ex1)
                {
Logger.getLogger(GameEngine.class.getName()).log(java.util.logging.Level.SEVERE,null,ex1);

                }
```

MOVE FUNCTION FOR BIKE :

For each frame the move function is called for bike sprite where the movement happens based on previous selected direction with the constant velocity *(BIKE_MOVEMENT)*
Move function calls moveX and moveY function which are adds the value to the x and y coordinates based on their velx and vely values

```
 /**
	* Moves the sprite Bike in x direction also checks the bound as well and
if true then change isAlive status to false
	*/
	public void moveX() {
		x += velx;
		if (x + width >= 800 || x <= 0) {
		//return false;
		isAlive=false;
		}
		if(velx!=0){
		addTrace();
		}
	}
```

moveY is same implementation logic

AddTrace() function basically is called each time when in move which create LightTrace instance to add in ArrayList of bikes trail and also check if its touched its own trace

```
/*** add a new coOrdinates (x,y) in the LightTrace instance array with same
size*/
	public void addTrace()
	{
	int a = x;
	int b = y;
	LightTrace t=new LightTrace(a, b, LightTrace.size,
LightTrace.size,this.getColor());
	ArrayList<LightTrace> pa = this.getPath();
	// pa.forEach((x)->System.out.println("lettssee "+x.collides(t)));
	for(LightTrace p:pa)
	{
	if(p.getX1()==t.getX1() && p.getY1()==t.getY1())
		this.isAlive=false;
	}
	if(!pa.contains(t))
	{			trail.add(t);	}}
```

DRAW FUNCTIONS :

Basic draw function implementation in Sprite

```
/**
 * draws the image file of sprite
 * @param g
 */
      public void draw(Graphics g) {
      g.drawImage(image, x, y, width, height, null);
      }
```

Draw function implementation in LighTrace :

```
  /**
      * draws the line
      * @param gc
      */
      public void draw(Graphics gc) {
            gc.drawRect(this.x, this.y, this.width, this.height);
      }
```

Draw Function implementation in BikeClass:

```
public void draw(Graphics g){
      super.draw(g);
      for (LightTrace k: trail)
       {  k.draw(g); }   }
```

EVENT HANDLING :

Each keystroke functionality is mapped in gameEngine constructor which changes the velocity of players based on button pressed.

```
/**
	* checks the keystrokes and takes corresponding actionPerformed and  creates instance of newFrameListener
	*/

public GameEngine() {
	super();
	if(ChoseRounds!=0)  levelNum=ChoseRounds;
	background = new ImageIcon("src/res/Data/background2.png").getImage();

	this.getInputMap().put(KeyStroke.getKeyStroke("LEFT"), "pressed left");
	this.getActionMap().put("pressed left", new AbstractAction() {
		@Override
		public void actionPerformed(ActionEvent ae) {

		player1.setVelx(-MOTOR_VELOCITY);
		}
	});
	this.getInputMap().put(KeyStroke.getKeyStroke("RIGHT"), "pressed right");
	this.getActionMap().put("pressed right", new AbstractAction() {
		@Override
		public void actionPerformed(ActionEvent ae) {

		player1.setVelx(MOTOR_VELOCITY);
		}
	});
	this.getInputMap().put(KeyStroke.getKeyStroke("DOWN"), "pressed down");
	this.getActionMap().put("pressed down", new AbstractAction() {
		@Override
		public void actionPerformed(ActionEvent ae) {

		player1.setVely(MOTOR_VELOCITY);
		}
	});
	this.getInputMap().put(KeyStroke.getKeyStroke("UP"), "pressed up");
	this.getActionMap().put("pressed up", new AbstractAction() {
		@Override
		public void actionPerformed(ActionEvent ae) {

		player1.setVely(-MOTOR_VELOCITY);
		}
	});

	this.getInputMap().put(KeyStroke.getKeyStroke("A"), "pressed d");
	this.getActionMap().put("pressed d", new AbstractAction() {
		@Override
```

```
            public void actionPerformed(ActionEvent ae) {
            player2.setVelx(-MOTOR_VELOCITY);
            }
      });
      this.getInputMap().put(KeyStroke.getKeyStroke("D"), "pressed a");
      this.getActionMap().put("pressed a", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent ae) {
            player2.setVelx(MOTOR_VELOCITY);
            }
      });
      this.getInputMap().put(KeyStroke.getKeyStroke("S"), "pressed s");
      this.getActionMap().put("pressed s", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent ae) {
            player2.setVely(MOTOR_VELOCITY);
            }
      });
      this.getInputMap().put(KeyStroke.getKeyStroke("W"), "pressed w");
      this.getActionMap().put("pressed w", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent ae) {

            player2.setVely(-MOTOR_VELOCITY);
            }
      });
      this.getInputMap().put(KeyStroke.getKeyStroke("ESCAPE"), "escape");
      this.getActionMap().put("escape", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent ae) {
            paused = !paused;
            }
      });
      restart();
      newFrameTimer = new Timer(1000 / FPS, new NewFrameListener());
      newFrameTimer.start();
      }
```

HIGHSCORE DATABASE :

When a game is over endgame() function is called which find the maximum of of score and it to the highscore table with winner name

```
 public void endGame()
{

      JOptionPane.showMessageDialog(this, " FinalScores!\n Name: " +
player1.getName() + " Score : " +score1.toString()+
```

```
                                                "\nName: "+ player2.getName()+"
Score : "+ score2.toString());

        String winnerName="" ;
        Integer winnerScore=0;
        if(score1>score2)
        {
                winnerName=player1.getName();
                winnerScore=score1;
        }else if(score1<score2) {
                winnerName=player2.getName();
                winnerScore=score2;
        }



        try {

                HighScoreDB highScores = new HighScoreDB(10);
                if(winnerScore>0)
                highScores.putHighScore(winnerName, winnerScore );
                System.out.println(highScores.getHighScoreDB());

        } catch (SQLException ex1) {


Logger.getLogger(GameEngine.class.getName()).log(java.util.logging.Level.SEVERE,
null, ex1);

        }
        score1=0;
        score2=0;
        }
```