PingPong

Patrick Stockton (pyw282)
EE 5123 – Engineering Programming II
Electrical & Computer Engineering Department
The University of Texas at San Antonio, College of Engineering
San Antonio, Texas
pstockton@ieee.org

***Abstract*—The Java class "PingPong" is an interactive simulation of a ping pong game using the Java graphics libraries, where the user can control a paddle, and deflect the ping pong ball against an AI controlling the opposing paddle. The class displays the objects in the simulation on a panel, and tracks all data values within the runtime.**

Keywords—Java; simulation; PingPong; game; graphics

I. INTRODUCTION (*HEADING 1*)

Videogames have played a huge role in Java and applet design over the recent years, due to the large consumption of entertainment and advancement in technology. This "PingPong" Java class enables a user to interact with a simulation made from the graphics portion of Java by allowing the user to control a paddle using pre-determined keystrokes, and competing against an artificial intelligent (AI) controlled paddle. The goal of the simulation is to deflect the ping pong ball past either paddle to score, and reach an end score to win.

This Java class utilizes the input/output control library by allowing the user to edit specific settings of the game before it is loaded from a configuration file named "config.txt". This allows the user to customize how the simulation will be generated, by providing values such as "end score", "player name", "background color", etc.

Once the configuration file is scanned and inputted, the user can then begin playing until the end score is reached by either the AI or the user.

II. SPECIFICATIONS

The specifications of the PingPong class comprise of the constructors, methods, and data type members that create the functioning simulation.

A. Constructors

Constructor	*Parameters*	*Use*
startGame	DrawingPanel, Graphics	Used for running the simulation
drawBall	Graphics, Color	Creates the ball graphic
drawPaddle	Graphics, Color	Creates the user paddle
drawPaddleAI	Graphics, Color	Creates the AI paddle

A. Data Members

Data Member	*Type*	*Use*
PANEL_WIDTH	final int	Set static panel width
PANEL_HEIGHT	final int	Set static panel height
BALL_SIZE	final int	Set the initial ball size
BALL_COLOR	final graphic	Set the initial ball color
ballX	int	The ball's X location
ballY	int	The ball's Y location
BACKGROUND_COLOR	Final graphic	Set the background Color
ballVelocityX	Int	Ball change in X speed
ballVelocityY	Int	Ball change in Y speed
KEY_SPACE	Final int	Integer constant for "space key" identification
PADDLE_LENGTH	Final int	Length of paddle
AI_paddle_length	Final int	Length of AI paddle
PADDLE_X	Final int	Static X location of user paddle
AI_paddleX	Final int	Static X location of AI paddle
paddleY	Int	Initial Y location of user paddle
AI_paddleY	Int	Initial Y location of AI paddle
PADDLE_COLOR	Final graphic	Color of paddle
UP_ARROW	Final int	Integer constant for "up arrow" identification
DOWN_ARROW	Final int	Integer constant for "down arrow"

		identification
AI_score	Int	Score of AI
USER_score	Int	Score of the user
Targetscore	Int	End score limit
Go	Boolean	Run the class
Score_Color	Final graphic	Color of the score text
normalFont	Final graphic	Size of the default font
scoreFont	Final graphic	Size of the score font

B. *Methods*

Method	*Parameters*	*Use*
readConfig	String[] args	Opens and reads the config.txt file's values
moveBall	Graphics	Determines the new coordinates of drawing the ball
resetBall	Graphics	Sets and draws ball's coordinates at origin
handleKeys	DrawingPanel, Graphics	Tests keycodes through the keylistener
movePaddle	Graphics, Int	Calls the drawPaddle and alters paddle coordinates
movePaddleAI	Graphics, Int	Calls the drawPaddleAI and alters paddle coordinates
detectHit	Void	Tests if ball location is the same as either paddle
displayScore	Graphics, Color	Draws score values on panel
randomGen	Void	Computes random values for ball velocity

III. CLASS

A. *The approach of the design of the Class**

Upon conducting research of approaches of the project, it was discovered that a course in the Department of Computer Science at UTSA required the students of the course to develop a ping pong game simulation throughout the 16 weeks of the semester. The instructor of this course, **Steven Robbins**, provided a structured approach on completing the project, which is the approach that was taken in this project.

As the time frame of completing this project for EE 5123 was only a few weeks compared to the 16-weeks of the original course, it offered a challenge. Thus, I had chosen to follow the original approach that was offered on the course website (link provided below), where I used the variable names recommended on the online platform, **but all code was created myself**. The additional Java class used in this project, "DrawingPanel", was included by the course professor, which is gone into more detail in *III.B.* below.

To add a customized twist to the project, I included the use of the I/O library to read a configuration file that the user can edit to supply different/desired values upon compilation and runtime.

http://www.cs.utsa.edu/~cs1063/

B. *DrawingPanel Class*

The "*DrawingPanel*" Java class was a supporting resource that was provided through Dr. Robbins on the courseware. This "*DrawingPanel*" class's basic function is to provide the core functionality of the "Keylistener" methods of Java, which allow for the use of key strokes while running the program. The implementation of the Keylistener was essential in order for the simulation to work correctly, and it was conveniently designed via the supported class.

Note: I did not write the "DrawingPanel" Class, as it was supplied via the link displayed above for the course's material.

C. *How to use the "PingPong" Class*

The design of the "*PingPong*" class was done through the **Dr. Java** IDE platform. In order to successfully compile and run the class, you must possess three items:

- "PingPong.java"
- "DrawingPanel.java"
- "Config.txt"

The location of the "Config.txt" file must be entered in the "PingPong.java" class on *line 83*, seen below:

```
82   public static void readConfig(String[] args) throws Exception {
83     String infile = "C:\\Users\\patri\\Desktop\\PingPong\\Config.txt";
```

Figure 1: "Config.txt" Location

When the "PingPong.java" and "DrawingPanel.java" are in the common folder, and the target location of the "Config.txt" file is corrected, then you may compile the program.

D. *"Config.txt" File*

The "Config.txt" file allows for the customization of the simulation prior to runtime. This file includes values that can be edited by the user with their desired values.

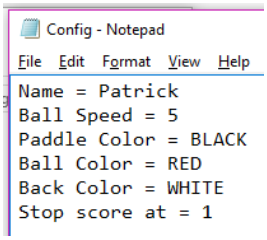

Figure 2: "Config.txt" File Contents

These values effect the simulation differently.

Line 1: Desired user name

Line 2: Initial ball speed

Line 3: Color of paddles

Line 4: Color of the ball

Line 5: Background color

Line 6: Ending score value

E. Features of the "PingPong" Class

The "*PingPong"* class offers various values that can be manipulated during the runtime of the program.

First step is to launch the simulation. Below is the default starting values of the graphical display.

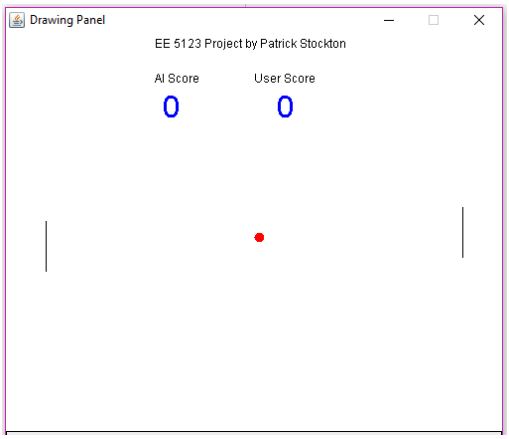

Figure 3: Launch Panel

The default panel width and height is 500x400 respectively. This creates the panel window seen above. The default locations for the ball, user paddle, AI paddle, and score are all set.

In order to begin the simulation/game, the key stroke of the **space bar** must be entered (hit the space bar). This will trigger for the simulation to begin.

In early development, the "*randomGen( )*" method is to allow for the ball to begin traveling in random trijectories from the original, either during the "*resetBall( )*" method call or the "*startGame( )*" constructor. Unfortunately, this method was not functioning correctly in the end.

Once the space bar key stroke is entered, the simulation begins. The "*keyHandler( )"* method utilizes the "DrawingPanel" class's key listener to determine which key is pressed and released. This allows for the correct ASCII value to be determined.

F. Detecting Ball Collision

During the runtime of the simulation, it is necessary to determine if the ball collides with any determined obstacle in the game.

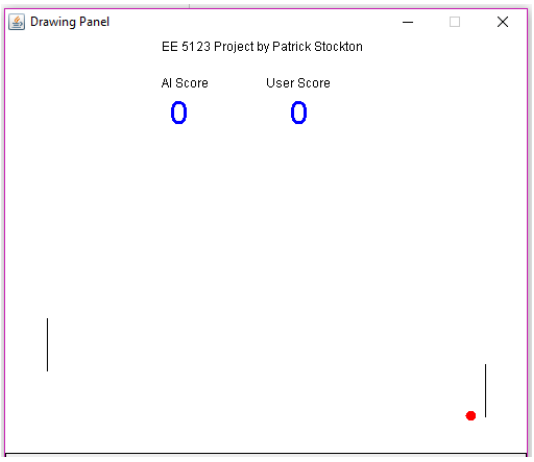

Figure 4: Ball collision

The "*detectHit( )*" method determines if the ball has collided with either the user's paddle, the AI's paddle, the top panel boundary, or the bottom panel boundary. This method changes the ball's X and Y trajectory depending on which boundary has been hit. As seen in **Figure 4**, the ball will change it's X trajectory (negative to positive, and positive to negative) if a paddle is collided with.

In this scenario, the ball is traveling to the right of the screen (positive X, negative Y), upon colliding with the user's paddle, the ball trajectory is changed to (negative X, positive Y), which propels the ball to the top left of the screen.

The paddle itself offers different values in which to increase or decrease the speed of the ball after collision. This speed change is determined on which area that the ball collides with the paddle as seen in the figure below:

Figure 5: Paddle Collision Areas

G. *Scoring*

Score is determined if the ball has passed either the user's or AI's score boundary, which is the edge of the panel. If the ball crossed the right hand panel boundary, the user will gain a point, otherwise if the ball crosses the left hand panel boundary, the AI will gain a point.

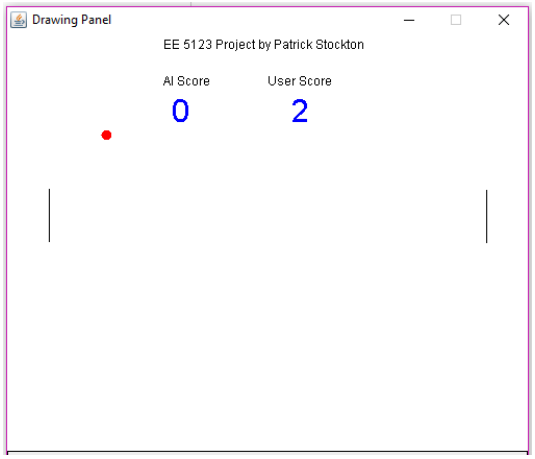

Figure 6: Scoring

Once the ball has crossed one of the boundaries, the "*resetBall( )*" method is called which resets the ball to the origin.

H. *Ending the game*

To determine the winner of the game, either the user's or AI's score must reach the target score, or the "endscore" value. The end score can be set in the "Config.txt" file by entering an integer value to determine what the max score will be.

If either the AI or user's score reaches that end score value, the game is over and a display message is displayed on the screen.

Figure 7: User Wins

Figure 8: AI Wins

IV. Conclutions

The design process provided a few challenges and design problems that were encountered. An initial issue was inputting the values read from the "Config.txt" file, and saving them as global variables that would be read into the runtime functions. This was because of the "static to non-static" variable casting, and was resolved by creating an object that would reference the read data, which can be seen below:

```
// Reading the data files from the config.txt file
try {
  PingPong obj = new PingPong();
  obj.readConfig(args); // Reading config file
} catch (Exception e) {
  e.printStackTrace();
}
```

Apart from the I/O challenges, the rest revolved around the interaction on the panel with various objects. For example; if I changed the ball speed by an amount such as 15 (the default

speed is 4), the ball would move across the screen in increments of 15. The issues arises if the ball is 13 pixels away from the paddle, and performs another move, it would be 2 pixels past the paddle and a collision would not be detected. This would result in the ball "ghosting" through the paddle and into the score zone.

Apart from issues with the coding process, the greatest one arose with file management. I was updating the project code on my laptop (online), as well as testing it on my tablet (offline). These project files were saved in a common folder on Dropbox. I updated the file on my laptop (which was online), and added a new file from my tablet to the same folder (the tablet was offline). After noticing the tablet was offline, I connected to the internet, and unfortunately, the tablet upload overwrite the original file, thus I lost my progress. This being the case, I had to spend the last 48 hours completely rebuilding the project (a couple very long nights). Lesson to be learned: have many backups (local and cloud).

The overall project was very interesting and provided new insights. And as I mentioned in the above section, it was based on the CS course project at UTSA which has a much longer time scale, therefore it was crunch time for these last couple weeks.

REFERENCES

The template will number citations consecutively within brackets [1]. The sentence punctuation follows the bracket [2]. Refer simply to the reference number, as in [3]—do not use "Ref. [3]" or "reference [3]" except at the beginning of a sentence: "Reference [3] was the first ..."

[1] Steven Robbins, UTSA CS-1063 course, www.cs.utsa.edu/~cs1063/

[2] Trevor Page, Casting in Java, howtoprogramwithjava.com.

APPENDIX I – PINGPONG.JAVA CODE

```
//////////////////////////////////////////////////
//    EE 5123 - Final Project
//
//    By: Patrick Stockton
//       @01291745   pyw282
//
//    Class: Ping Pong
//////////////////////////////////////////////////

import java.io.*;
import java.awt.*;
import java.util.*;

public class PingPong {
  public static final int PANEL_WIDTH = 500; // "Width" global constant variable
  public static final int PANEL_HEIGHT = 400; // "Height" global constant variable
  public static final int SLEEP_TIME = 50; // "Sleep" global constant variable
  public static final int BALL_SIZE = 10; // "Ball Size" global constant variable
  public static final Color BALL_COLOR = Color.RED; // "Ball Color" global constant variable
  public static int ballX, ballY; // Ball coordinates instantiation, global class variables
  public static final Color BACKGROUND_COLOR = Color.WHITE; // "Background Color" global constant variable
  public static int ballVelocityX, ballVelocityY; // Ball moving global class variables
  public static final int KEY_SPACE = ' '; // Integer constant for "space key" identification
  public static final int PADDLE_LENGTH = 50; // Length of Paddle
  public static final int AI_paddle_length = 50; // Length of computer's paddle
  public static final int PADDLE_X = 460; // Paddle stays 40 pixels from right side of panel
  public static final int AI_paddleX = 40; // Starting location for computer's paddle on panel
  public static int paddleY; // Initilization of Paddle's Y coordinate
  public static int AI_paddleY;
  public static final int AI_paddle_maxspeed = 2;
  public static final Color PADDLE_COLOR = Color.BLACK; // Paddle's Color
  public static final int UP_ARROW = 38; // Integer constant for "Up Arrow key" identification
  public static final int DOWN_ARROW = 40; // Integer constant for "Down Array key" identification
  public static final int PADDLE_MOVE_AMOUNT = 5; // Integer for the amount the paddle moves
  public static int init_ball_velx = 4;
  public static int init_ball_vely = 3;
  public static int AI_score, USER_score, targetscore; // Scores for both the AI and user, and target score
  public static boolean go = true;
  public static final Color Score_Color = Color.BLUE; // Score Color
  public static Font normalFont;
  public static Font scoreFont;
  public static final int SCORES_FONT_SIZE = 30;
  public static final int MIN_X_VELOCITY = -2, MAX_X_VELOCITY = 4; // Starting min and max X velocity of the ball
  public static final int MIN_Y_VELOCITY = -2, MAX_Y_VELOCITY = 3; // Starting min and max Y velcoity of the ball

  public int ballspeed = 0;

  public static void main(String[] args) {

    // Reading the data files from the config.txt file
    try {
      PingPong obj = new PingPong();
      obj.readConfig(args); // Reading config file
    } catch (Exception e) {
      e.printStackTrace();
    }
    //readConfig(args);
      DrawingPanel panel = new DrawingPanel(PANEL_WIDTH, PANEL_HEIGHT);
        Graphics g = panel.getGraphics( );
        g.drawString("EE 5123 Project by Patrick Stockton", 150, 15); // Print name on screen/panel
        g.drawString("AI Score", 150, 50);
        g.drawString("User Score", 250, 50);

        ballX = 250; // Ball origin coordinates
        ballY = 200; // Ball origin coordinates
        ballVelocityX = 0; // Initial ball velocity (X)
        ballVelocityY = 0; // Initial ball velocity (Y)
        paddleY = 175; // Set paddle at center of Y axis
        AI_paddleY = 175; // Set AI paddle at center of Y axis

        drawBall(g, BALL_COLOR); // Ball drawing call
        drawPaddle(g, PADDLE_COLOR);

        normalFont = g.getFont();
        scoreFont = new Font(normalFont.getName(), normalFont.getStyle(), SCORES_FONT_SIZE);

        startGame(panel, g);
  } // end of main


  // Reads the configuration file and imports settings
  public static void readConfig(String[] args) throws Exception {
    String infile = "C:\\Users\\patri\\Desktop\\PingPong\\Config.txt";
    String line;
    String username = "", backgroundcolor = "";
    int ballspeed2 = 0, endscore = 0;

    // Read from file
    try {
     FileReader freader = new FileReader(infile);
     BufferedReader in = new BufferedReader(freader);

     // Scan text file
     while((line = in.readLine()) != null) {
       System.out.println(line);
```

```
            if (((line.substring(0,4)).equals("Name"))) {
              username = line.substring(7); // begin at start of name entry field
            }
            if (((line.substring(0,10)).equals("Ball Speed"))) {
              ballspeed2 = Integer.parseInt(line.substring(13)); // convert to int
            }
            if (((line.substring(0,10)).equals("Back Color"))) {
              backgroundcolor = line.substring(13); // background color
            }
            if (((line.substring(0,4)).equals("Stop"))) {
              targetscore = Integer.parseInt(line.substring(16)); // target end score for round/game to end
            }
           }

           in.close(); // close config file
           System.out.println("EOF Reached!");
           }

           catch(IOException f) {
            System.out.println("ERROR Found " + f);
           }

           System.out.println("User name is: " + username);
           System.out.println("Ball Speed is: " + ballspeed2);
           System.out.println("Background Color is: " + backgroundcolor);
           System.out.println("End Score Goal is: " + endscore + " points!");
          } // end of readConfig


         // Function for running/refreshing the game
         public static void startGame(DrawingPanel panel, Graphics g) {
           int x = 0;
           int y = 270;
           int deltaX = 1;
           int deltaY = 0;
           //targetscore = 2;
           //boolean go = true;

           while (go == true) {
             handleKeys(panel, g);
               panel.sleep(50);
               moveBall(g);
               detectHit( );
               movePaddle(g, deltaY);
               movePaddleAI(g, deltaY);
               //displayScore(g, Score_Color);
           }
          } // end of startGame


         // Function for creating a ball of color "c"
         public static void drawBall(Graphics g, Color c) {
           g.setColor(c);
           g.fillOval(ballX, ballY, BALL_SIZE, BALL_SIZE);

         } // end of drawBall


         // Function for moving the ball
         public static void moveBall(Graphics g) {
           drawBall(g, BACKGROUND_COLOR); // erase ball
           displayScore(g, BACKGROUND_COLOR); // Score erase
           ballX = ballX + ballVelocityX; // set new X coordinate (increment)
           ballY = ballY + ballVelocityY; // set new Y coordinate (increment)
           drawBall(g, BALL_COLOR);
           displayScore(g, Score_Color); // Draw score color

           if (ballX < -10) {
             displayScore(g, BACKGROUND_COLOR);
             USER_score++;
             drawBall(g, BACKGROUND_COLOR);
             resetBall(g);
           }
           if (ballX > 500) {
             displayScore(g, BACKGROUND_COLOR);
             AI_score++;
             drawBall(g, BACKGROUND_COLOR);
             resetBall(g);
           }
         } // end of moveBall


         // Function for resetting the ball
         public static void resetBall(Graphics g) {
           drawBall(g, BACKGROUND_COLOR);
           ballX = 250;
           ballY = 200;
           //randomGen();
           ballVelocityX = init_ball_velx; // Default speed = 4
           ballVelocityY = init_ball_vely; // Default speed = 1
           drawBall(g, BALL_COLOR);

         } // end of resetBall


         // Function for receiving keyboard inputs
         public static void handleKeys(DrawingPanel panel, Graphics g) {
           int keyCode = panel.getKeyCode( );
           if (keyCode == KEY_SPACE)
             resetBall(g);
           else if (keyCode == UP_ARROW)
             movePaddle(g, -PADDLE_MOVE_AMOUNT);
           else if (keyCode == DOWN_ARROW)
             movePaddle(g, PADDLE_MOVE_AMOUNT);

         } // end of handleKeys


         // Function for creating the Paddle
         public static void drawPaddle(Graphics g, Color c) {
           g.setColor(c);
           g.drawLine(PADDLE_X, paddleY, PADDLE_X, paddleY + PADDLE_LENGTH);
         } // end of drawPaddle


         // Function for creating the AI's Paddle
         public static void drawPaddleAI(Graphics g, Color c) {
           g.setColor(c);
           g.drawLine(AI_paddleX, AI_paddleY, AI_paddleX, AI_paddleY + AI_paddle_length);
         } // end of drawPaddle


         // Function for moving the Paddle
         public static void movePaddle(Graphics g, int deltaY) {
           drawPaddle(g, BACKGROUND_COLOR);

           if (paddleY < 400 && paddleY > 0)
             paddleY = paddleY + deltaY;
           if (paddleY >= 380)     // Prevent paddle from going off the bottom of the panel
             paddleY = paddleY - 1;   // If paddle is moving off bottom panel, push up by 1
           if (paddleY <= 0)       // Prevent paddle from going off the top of the panel
             paddleY = paddleY + 1;   // If paddle is moving off top panel, push down by 1

           drawPaddle(g, PADDLE_COLOR);
           //drawPaddleAI(g, PADDLE_COLOR);

         } // end of movePaddle


         // Function for moving the AI's Paddle
         public static void movePaddleAI(Graphics g, int deltaY) {
           drawPaddleAI(g, BACKGROUND_COLOR);
           //deltaY = 5;
           if (AI_paddleY < ballY - 10)
             AI_paddleY = AI_paddleY + AI_paddle_maxspeed;
           if (AI_paddleY >= ballY - 10)       // Prevent paddle from going off the bottom of the panel
             AI_paddleY = AI_paddleY - AI_paddle_maxspeed;    // If paddle is moving off bottom panel, push up by 1
           //if (paddleY <= 0)       // Prevent paddle from going off the top of the panel
             //paddleY = paddleY + 1;   // If paddle is moving off top panel, push down by 1

           drawPaddleAI(g, PADDLE_COLOR);

         } // end of movePaddleAI


         // Function for detecting collisions
         public static void detectHit( ) {

           // Paddle collision detection
```

```
        //if (ballVelocityX > 0 && (ballY >= paddleY - 10 && ballY <= paddleY +
PADDLE_LENGTH) && ballX == PADDLE_X - 10)
          //ballVelocityX = -ballVelocityX;
        if (ballVelocityX > 0 && (ballY >= paddleY - 5 && ballY <= paddleY +
PADDLE_LENGTH - 50) && ballX == PADDLE_X - 10)
          ballVelocityX = -9;
        if (ballVelocityX > 0 && (ballY >= paddleY && ballY <= paddleY +
PADDLE_LENGTH - 44) && ballX == PADDLE_X - 10)
          ballVelocityX = -8;
        if (ballVelocityX > 0 && (ballY >= paddleY && ballY <= paddleY +
PADDLE_LENGTH - 38) && ballX == PADDLE_X - 10)
          ballVelocityX = -7;
        if (ballVelocityX > 0 && (ballY >= paddleY && ballY <= paddleY +
PADDLE_LENGTH - 32) && ballX == PADDLE_X - 10)
          ballVelocityX = -6;
        if (ballVelocityX > 0 && (ballY >= paddleY && ballY <= paddleY +
PADDLE_LENGTH - 26) && ballX == PADDLE_X - 10)
          ballVelocityX = -5;
        if (ballVelocityX > 0 && (ballY >= paddleY && ballY <= paddleY +
PADDLE_LENGTH - 20) && ballX == PADDLE_X - 10)
          ballVelocityX = -4;
        if (ballVelocityX > 0 && (ballY >= paddleY && ballY <= paddleY +
PADDLE_LENGTH - 14) && ballX == PADDLE_X - 10)
          ballVelocityX = -3;
        if (ballVelocityX > 0 && (ballY >= paddleY && ballY <= paddleY +
PADDLE_LENGTH - 8) && ballX == PADDLE_X - 10)
          ballVelocityX = -2;
        if (ballVelocityX > 0 && (ballY >= paddleY && ballY <= paddleY +
PADDLE_LENGTH) && ballX == PADDLE_X - 10)
          ballVelocityX = -1;

        if (ballY >= 400 || ballY <= 0) // Top and Bottom boundary collision detection
          ballVelocityY = -ballVelocityY;
        if (ballVelocityX < 0 && (ballY >= AI_paddleY && ballY <= AI_paddleY +
AI_paddle_length) && ballX == AI_paddleX - 10)
          ballVelocityX = 4;

       } // end of detectHit


       // Display scores
       public static void displayScore(Graphics g, Color col) {
         g.setColor(col);
         g.setFont(scoreFont);
         g.drawString(" " + AI_score, 150, 85);
         g.drawString(" " + USER_score, 265, 85);

         // Test to see if the final score is reached
         if (AI_score == targetscore) {
          g.drawString("GAME OVER!", 150, 200);
          g.drawString("AI has won!", 150, 250);
          go = false;
         }
         else if (USER_score == targetscore) {
          g.drawString("GAME OVER!", 150, 200);
          g.drawString("User has won!", 150, 250);
          go = false;
         }
       } // end of displayScore


       // Generate random velocities
       public static void randomGen( ) {
         Random rand = new Random();

         init_ball_velx = MIN_X_VELOCITY + rand.nextInt(MAX_X_VELOCITY);
         init_ball_vely = MIN_Y_VELOCITY + rand.nextInt(MAX_Y_VELOCITY);
         System.out.println("init_ball_velx = " + init_ball_velx);
         System.out.println("init_ball_vely = " + init_ball_vely);
         System.out.println("========");
       } // end of randomGen

      } // end of class
```

APPENDIX II – DRAWINGPANEL.JAVA CODE

```
/*
Stuart Reges and Marty Stepp
February 24, 2007
Changes by Tom Bylander in 2010 (no anti-alias, repaint on sleep)
Changes by Tom Bylander in 2012 (track mouse clicks and movement)
Changes by Tom Bylander in 2013 (fix bug in tracking mouse clicks)
Changes by S. Robbins   in 2014 (getters for width and height)
Changes by S. Robbins   in 2014 (addKeyListener added)
Changes by S. Robbins   in 2014 (catch exception on default close so that it works in an applet)
Changes by S. Robbins   in 2015 (buffer key events)
Changes by S. Robbins   in 2015 (show mouse status by default is off)

The DrawingPanel class provides a simple interface for drawing persistent
images using a Graphics object.  An internal BufferedImage object is used
to keep track of what has been drawn.  A client of the class simply
constructs a DrawingPanel of a particular size and then draws on it with
the Graphics object, setting the background color if they so choose.

To ensure that the image is always displayed, a timer calls repaint at
regular intervals.
*/

import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import javax.swing.*;
import javax.swing.event.*;
import java.util.ArrayList;

public class DrawingPanel implements ActionListener {
 private static final String versionMessage =
    "Drawing Panel version 1.1, January 25, 2015";
 private static final int DELAY = 100;  // delay between repaints in millis
 private static final boolean PRETTY = false;  // true to anti-alias
 private static boolean showStatus = false;
 private static final int MAX_KEY_BUF_SIZE = 10;

 private int width, height;    // dimensions of window frame
 private JFrame frame;         // overall window frame
 private JPanel panel;         // overall drawing surface
 private BufferedImage image;  // remembers drawing commands
 private Graphics2D g2;        // graphics context for painting
 private JLabel statusBar;     // status bar showing mouse position
 private volatile MouseEvent click;     // stores the last mouse click
 private volatile boolean pressed;      // true if the mouse is pressed
 private volatile MouseEvent move;      // stores the position of the mouse
 private ArrayList<KeyInfo> keys;

 // construct a drawing panel of given width and height enclosed in a window
 public DrawingPanel(int width, int height) {
  this.width = width;
  this.height = height;
  keys = new ArrayList<KeyInfo>();
  image = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);

  statusBar = new JLabel(" ");
  statusBar.setBorder(BorderFactory.createLineBorder(Color.BLACK));
  statusBar.setText(versionMessage);

  panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 0, 0));
  panel.setBackground(Color.WHITE);
  panel.setPreferredSize(new Dimension(width, height));
  panel.add(new JLabel(new ImageIcon(image)));

  click = null;
  move = null;
  pressed = false;

  // listen to mouse movement
  MouseInputAdapter listener = new MouseInputAdapter() {
   public void mouseMoved(MouseEvent e) {
    pressed = false;
    move = e;
    if (showStatus)
     statusBar.setText("moved (" + e.getX() + ", " + e.getY() + ")");
   }

   public void mousePressed(MouseEvent e) {
    pressed = true;
    move = e;
    if (showStatus)
     statusBar.setText("pressed (" + e.getX() + ", " + e.getY() + ")");
   }

   public void mouseDragged(MouseEvent e) {
    pressed = true;
    move = e;
    if (showStatus)
     statusBar.setText("dragged (" + e.getX() + ", " + e.getY() + ")");
   }

   public void mouseReleased(MouseEvent e) {
    click = e;
    pressed = false;
    if (showStatus)
     statusBar.setText("released (" + e.getX() + ", " + e.getY() + ")");
   }

   public void mouseEntered(MouseEvent e) {
//    System.out.println("mouse entered");
    panel.requestFocus();
   }

  };
  panel.addMouseListener(listener);
  panel.addMouseMotionListener(listener);
  new DrawingPanelKeyListener();

  g2 = (Graphics2D)image.getGraphics();
  g2.setColor(Color.BLACK);
  if (PRETTY) {
   g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
   g2.setStroke(new BasicStroke(1.1f));
  }

  frame = new JFrame("Drawing Panel");
  frame.setResizable(false);
  try {
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);   // so that this works in an applet
  } catch (Exception e) {}
  frame.getContentPane().add(panel);
  frame.getContentPane().add(statusBar, "South");
  frame.pack();
  frame.setVisible(true);
  toFront();
  frame.requestFocus();

  // repaint timer so that the screen will update
  new Timer(DELAY, this).start();
 }

 public void showMouseStatus(boolean f) {
  showStatus = f;
 }

 public void addKeyListener(KeyListener listener) {
  panel.addKeyListener(listener);
  panel.requestFocus();
 }

 // used for an internal timer that keeps repainting
 public void actionPerformed(ActionEvent e) {
  panel.repaint();
 }

 // obtain the Graphics object to draw on the panel
 public Graphics2D getGraphics() {
  return g2;
 }

 // set the background color of the drawing panel
 public void setBackground(Color c) {
  panel.setBackground(c);
 }

 // show or hide the drawing panel on the screen
 public void setVisible(boolean visible) {
  frame.setVisible(visible);
 }
```

```
// makes the program pause for the given amount of time,
// allowing for animation
public void sleep(int millis) {
  panel.repaint();
  try {
    Thread.sleep(millis);
  } catch (InterruptedException e) {}
}

// close the drawing panel
public void close() {
  frame.dispose();
}

// makes drawing panel become the frontmost window on the screen
public void toFront() {
  frame.toFront();
}

// return panel width
public int getWidth() {
   return width;
}

// return panel height
public int getHeight() {
   return height;
}

// return the X position of the mouse or -1
public int getMouseX() {
  if (move == null) {
    return -1;
  } else {
    return move.getX();
  }
}

// return the Y position of the mouse or -1
public int getMouseY() {
  if (move == null) {
    return -1;
  } else {
    return move.getY();
  }
}

// return the X position of the last click or -1
public int getClickX() {
  if (click == null) {
    return -1;
  } else {
    return click.getX();
  }
}

// return the Y position of the last click or -1
public int getClickY() {
  if (click == null) {
    return -1;
  } else {
    return click.getY();
  }
}

// return true if a mouse button is pressed
public boolean mousePressed() {
  return pressed;
}

public synchronized int getKeyCode() {
  if (keys.size() == 0)
    return 0;
  return keys.remove(0).keyCode;
}

 public synchronized char getKeyChar() {
  if (keys.size() == 0)
    return 0;
  return keys.remove(0).keyChar;
}

 public synchronized int getKeysSize() {
   return keys.size();
 }

private synchronized void insertKeyData(char c, int code) {
  keys.add(new KeyInfo(c,code));
  if (keys.size() > MAX_KEY_BUF_SIZE) {
    keys.remove(0);
//    System.out.println("Dropped key");
  }
}

private class KeyInfo {
  public int keyCode;
  public char keyChar;

  public KeyInfo(char keyChar, int keyCode) {
    this.keyCode = keyCode;
    this.keyChar = keyChar;
  }
}

private class DrawingPanelKeyListener implements KeyListener {

  int repeatCount = 0;

  public DrawingPanelKeyListener() {
    panel.addKeyListener(this);
    panel.requestFocus();
  }

  public void keyPressed(KeyEvent event) {
//    System.out.println("key pressed");
    repeatCount++;
    if ((repeatCount == 1) || (getKeysSize() < 2))
      insertKeyData(event.getKeyChar(),event.getKeyCode());
  }

  public void keyTyped(KeyEvent event) {
  }

  public void keyReleased(KeyEvent event) {
    repeatCount = 0;
  }

}
```