Department of Electrical and Computer Engineering
ECSE 202 – Introduction to Software Development
Assignment 4
An implementation of the Game of Pong

Introduction

With completion of Assignment 3, you now have most of the elements necessary to implement a fully interactive game on your computer. This is best handled in two steps: i) following the same path as you did in Assignment 3, replace the left wall by a paddle object, called the Agent, that follows and returns the ball, and ii) add buttons to make the play interactive as shown below in Figure 1.

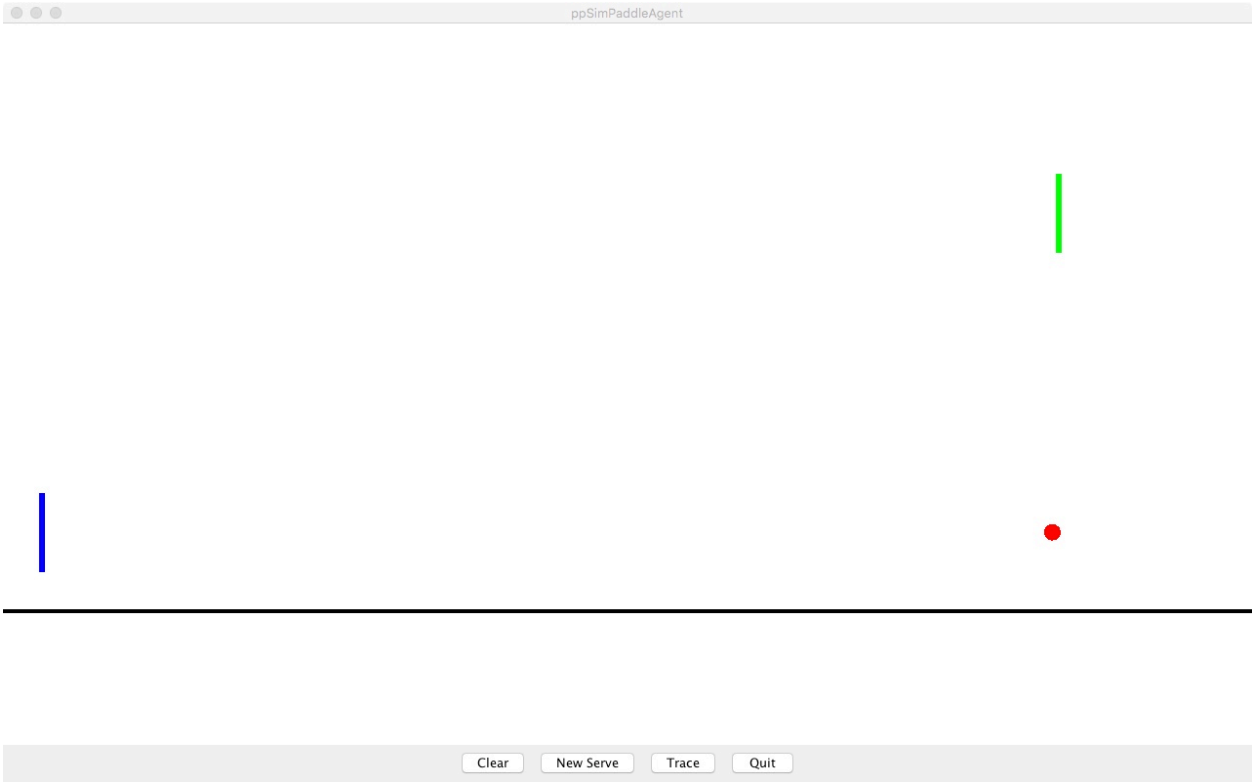

Figure 1

As in Assignment 3, play starts off with the Agent serving the ball from the left side of the screen and the Player returning it by using the mouse to control the paddle. Game play proceeds until either the Agent or the Player misses the ball. The New Serve button restores the game its initial state, while Quit exits the program. The Trace button enables trace points to be turned on or off (this is a toggle button), and Clear erases the display. You must implement this basic functionality in your program. (From here it is not difficult to add features such as a scoreboard and sliders to change the speed of the game and reaction time of the Agent).

Problem Description

Using the code from Assignment 3 as a starting point, write a new main class, ppSimPaddleAgent, that implements the scenario depicted in Figure 1. This will involve writing a new class, ppPaddleAgent, that extends ppPaddle by including a reactive element that adjusts the Y position of the Agent Paddle according to the Y position of the ball. Of course ppBall needs to be modified again as the left wall is replaced by the Agent Paddle, and ppTable needs some minor changes to reflect the change in layout. As with Assignment 3, we will be begin by describing the program classes:

1. The new entry point for the program is now called ppSimPaddleAgent with the following signature:

   ```
   public class ppSimPaddleAgent extends GraphicsProgram {
   }
   ```

   and has no explicit constructor. It is similar to the ppSimPaddle class from Assignment 3, but also enables ActionListeners and includes actionPerformed methods to implenent the simple user interface shown in Figure 1. The skeleton code found later in this document will outline the basic structure of this class.

2. The ppPaddle class has one minor change from Assignment 3, the addition of a new parameter to set the color of the paddle:

   ```
   public ppPaddle (double X, double Y, Color myColor, ppTable myTable) {
   }
   ```

3. The ppPaddleAgent class extends ppPaddle by including a provision for adjusting its Y position to match the ball within the run method. It has the identical constructor to its superclass:

   ```
   public ppPaddleAgent(double X, double Y, Color myColor, ppTable myTable) {
   }
   ```

   ppPaddleAgent should export the following new method:

public void attachBall(ppBall myBall)	Sets the value of the myBall instance variable in ppPaddleAgent.

4. A couple of changes need to be made to the ppBall class, one for functionality and the other to make the code a bit cleaner. This new version has the following constructor:

   ```
   public ppBall(double Xinit, double Yinit, double Vo, double theta, Color color, double
                 loss, ppTable table, boolean traceOn) {
   … (other code)
   }
   ```

The ppPaddle argument has been eliminated as references to both the Agent and Player paddles are set using the following new methods:

public void setPaddle (ppPaddle myPaddle)	Sets the value of the reference to the Player paddle.
public void setAgent (ppPaddleAgent theAgent)	Sets the value of the reference to the Agent paddle.

A new argument has been added to the end of the list, boolean traceOn, which enables trace points to be displayed when true.

An additional method is needed to determine if the ball is in play or not to prevent the user from starting a new game if one is already in progress. It is merely a getter for the control variable in the while loop.

public boolean ballInPlay()	A predicate that is true if a ppBall simulation is running.

Depending on how sophisticated your method for tracking the ball is in ppPaddleAgent, you will need getters for one or more of (X,Y,Vx,Vy).

5. The ppTable class requires a minor modification in the display setup as the left wall is now replaced by the Agent's paddle. Paddles should be drawn in the same color as the walls (Agent blue, Paddle green). There is no change to the constructor.

   The following additional method needs to be added:

void newScreen()	This method erases the current display by removing all objects and then regenerating the display; called whenever a new serve is requested.

6. There are some minor changes to ppSimParams:

i) *ppAgentXinit*	Initial X position of Agent paddle
ii) *ppAgentYinit*	Initial Y position of Agent paddle
iii) *ppAgentXgain*	Corresponding X gain for agent
iv) *ppAgentYgain*	Corresponding Y gain for agent
v) *ColorPaddle*	Player paddle color
vi) *ColorAgent*	Agent paddle color
vii) *ColorBall*	Ball color

```
// 1. Parameters defined in screen coordinates (pixels, acm coordinates)

    static final int scrWIDTH = 1080;          // n.b. screen coordinates
```

```
static final int scrHEIGHT = 600;
static final int OFFSET = 200;

// 2. Parameters defined in simulation coordinates (MKS, x-->range, y-->height)

static final double g = 9.8;                    // MKS
static final double k = 0.1316;                 // Vt constant
static final double Pi = 3.1416;
static final double XMAX = 2.74;                // Maximum value of X (pp table)
static final double YMAX = 1.52;                // Maximum value of Y (height above table)
static final double XLWALL = 0.1;               // position of left wall
static final double XRWALL = XMAX;              // position of right wall
static final double bSize = 0.02;               // pp ball radius
static final double bMass = 0.0027;             // pp ball mass
static final double XINIT = XLWALL+bSize;       // Initial ball location (X)
static final double TICK = 0.01;                // Clock tick duration (sec)
static final double ETHR = 0.001;               // Minimum ball energy needed to move
static final double Yinit = YMAX/2;             // Initial ball location (Y)
static final double PD = 1;                     // Trace point diameter
static final double SCALE = scrHEIGHT/YMAX;     // Pixels/meter
static final double ppPaddleH = 8*2.54/100;     // Paddle height
static final double ppPaddleW = 0.5*2.54/100;   // Paddle width
static final double ppPaddleXinit = XMAX-ppPaddleW/2;   // Initial Paddle X Position
static final double ppPaddleYinit = YINIT;      // Initial Paddle Y Position
static final double ppPaddleXgain = 2.0;        // Vx gain on paddle hit
static final double ppPaddleYgain = 2.0;        // Vy gain on paddle hit
static final double ppAgentXinit = XINIT;       // Initial Paddle X Position
static final double ppAgentYinit = YINIT;       // Initial Paddle Y Position
static final double ppAgentXgain = 2.0;         // Vx gain on paddle hit
static final double ppAgentYgain = 2.0;         // Vy gain on paddle hit
static final double TIMESCALE = 5000; // TICK to mSec scale factor (1000 -> real time)
static final Color ColorPaddle = Color.GREEN;   // User paddle color
static final Color ColorAgent = Color.BLUE;     // Agent paddle color
static final Color ColorBall = Color.RED;       // Ball color


// 3. Parameters used by the ppSim (main) class

static final int  NUMBALLS = 1;                 // # pp balls to simulate
static final double YinitMAX = 0.75*YMAX;       // Max inital height at 75% of range
static final double YinitMIN = 0.25*YMAX;       // Min inital height at 25% of range
static final double EMIN = 0.2;                 // Minimum loss coefficient
static final double EMAX = 0.2;                 // Maximum loss coefficient
static final double VoMIN = 5.0;                // Minimum velocity
static final double VoMAX = 5.0;                // Maximum velocity
static final double ThetaMIN = 0.0;             // Minimum launch angle
```

```
static final double ThetaMAX = 20.0;          // Maximum launch angle

// 4. Miscellaneous

static final boolean DEBUG = false;  // Enable debug messages and single step if true
static final boolean TEST = false;           // Print motion parameters
static final long RSEED = 8976232;           // Random number generator seed value
```

Classes in More Detail

The biggest change to the program occurs in the *ppSimPaddleAgent* class which now incorporates a primitive user interface that allows the Player to 1) Reset the screen, 2) Start over with a new Serve, 3) Enable/Disable trace points, and 4) Quit the game. A good starting point would be to create a method, newBall, with the following signature:

```
ppBall myBall = newBall() {
}
```

This method simply encapsulates generating random parameters and creating a ppBall instance, so that this process can be repeated each time a new serve is requested.

The template for ppSimPaddleAgent follows directly from ppSimPaddle:

```
public class ppSimPaddleAgent extends GraphicsProgram {
…declare instance variables…

    public void init() {

    …create Jbuttons for the 4 menu items…
    …add items to canvas…

     addMouseListeners();
     addActionListeners();

    …random number generator…

     myTable = new ppTable(this);
     myBall = newBall();

    …paddles…
     myPaddle = new ppPaddle(ppPaddleXinit,ppPaddleYinit,ColorPaddle,myTable);
     theAgent = new ppPaddleAgent(ppAgentXinit,ppAgentYinit,ColorAgent,myTable);

    …inform ball of paddle and agents…
```

```
        myBall.setPaddle(myPaddle);
        myBall.setAgent(theAgent);
        theAgent.attachBall(myBall);

        …add ball to display and start

        add(myBall.getBall());              // Each thread must be explicitly started
        theAgent.start();
        myPaddle.start();
        myBall.start();

    }                                       // End of init method
```

Calling init will set up the display (Figure 1) and start the initial play. This will continue until either the Agent or the Player misses the ball. From this point on, anything that happens will be the result of pushing a button.

The rest of this class is method definitions.

…mouse handler…

```
        public void mouseMoved(MouseEvent e) {
        myPaddle.setY(myTable.ScrtoY((double)e.getY()));
        }
```

…button handler…

```
        public void actionPerformed(ActionEvent e) {
                String command = e.getActionCommand();

                if (command.equals("Clear")) {
                …do clear…
                }

                else if  (command.equals("New Serve")) {
                …do new serve…
                }

                else if (command.equals("Quit")) {
                        System.exit(0);
                }
```

The Trace button is defined to be a toggle, which means it is set and cleared in the background without the need to decode its action event. If Trace is the instance variable corresponding to this JToggleButton, then we can simply pass Trace.isSelected() as an argument to the ppBall constructor to determine whether the trace is enabled or not.

The final method to be defined is newBall:

```
        ppBall newBall() {
        …generates parameters and creates an instance of ppBall.
        }
}
```

Most of the work in this assignment is here. The examples in the class slides provide all that you need to implement this program.

The *ppPaddleAgent* class and *ppPaddle* classes have the same constructor, which includes a minor change from Assignment 3:

```
public ppPaddleAgent(double X, double Y, Color myColor, ppTable myTable) {
…
}
```

A Color argument has been added so that paddles can be differentiated by color. There are two essential differences with ppPaddle:

i) The position of the paddle is explicitly updated on each iteration of the while loop in the run method by looking at the position and velocity of the ball. How you do this is at your discretion – the simplest (and least satisfying) approach is simply to set the Y position of the paddle to the Y position of the ball.

ii) ppPaddleAgent exports the attachBall method:

```
public void attachBall(pBall myBall) {
}
```

It is called when a new ball is created and used to set an instance variable in ppPaddleAgent that provides a reference to the ball. Position and velocity can then be accessed using ppBall's getter methods.

The *ppBall* class also merits some changes – the most obvious being replacing the left wall with the Agent paddle, exactly as you did for Assignment 3. You might also consider alternatives to how energy is added to the ball when hit with a paddle, e.g., Assignment 3, Part 5. The approach described there makes game play a bit more interesting by enabling the user to control the energy and direction of the ball.

Another problem is that both players can get caught in a "duel" where the energy keeps increasing without limit. An easy fix to this problem is to this problem is to use Math.min() to limit energy to some upper bound, e.g., the initial energy of the ball. Depending on your strategy, you may need to define additional parameters besides the ones listed earlier – just make sure that you include them in ppSimParams. Depending on how you implement the Agent's

code for intercepting the ball, you might need getters to retrieve the ball's position and velocity. At minimum, you will need a getter for Y.

In order to prevent starting a new serve while there is a ball already in play, a simple predicate method that returns the state of the control variable in the ppBall while loop is also needed:

public boolean ballInPlay() True if ball moving, false otherwise

And you will need to export to setters to allow ppBall access to paddle objects:

public void setPaddle (ppPaddle myPaddle)

public void setAgent (ppPaddleAgent theAgent)

Finally, *ppTable* is now reduced to drawing the ground plane, but exports an additional method:

public void newScreen() Erases screen, draws a new ground plane.

Instructions

1. Write/modify the Java classes described in this document: *ppSimPaddleAgent*, *ppPaddle*, ppPaddleAgent, *ppBall*, *ppTable* and *ppSimParams*. You can implement more classes if you wish, but you must implement the ones specified and any specified methods. Do this within the Eclipse environment so that it can be readily tested by the course graders; make sure that all of your classes are part of ppPackage. Your code must be fully documented.
2. Start your program with tracing on. As with Assignment 3, return the first ball with your paddle moving from top to bottom, and the second ball with the paddle moving from bottom to top. Do not return the third ball; capture the screen and save this file as A4-2.pdf (example shown below in Figure 2). A couple of details: i) when you create your JToggleButton, add true as the second argument to the constructor so that the program starts off with Trace enabled; ii) do not try to replicate the result shown in Figure 2 (the path will depend on the paddle). However you should be able to clearly see the two return paths with the final path terminating after the paddle.

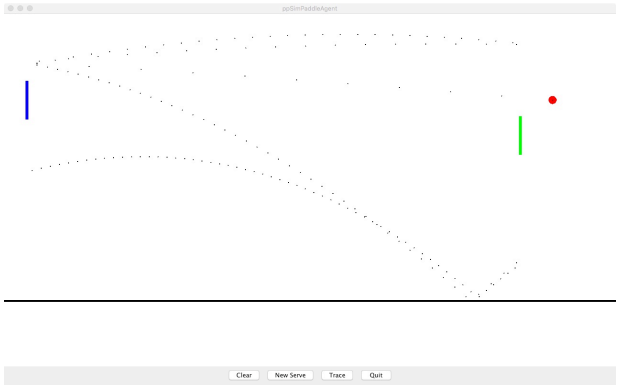

Figure 2

3. You may need to do some experimentation to find the right parameters to make your game "playable" – which means that you can go for several volleys with the Agent before missing the ball. Figure 3 shows an example of what this looks like for our implementation. Once you're satisfied with your game play, go a few rounds and capture the resulting screen; save this file as A4-3.pdf

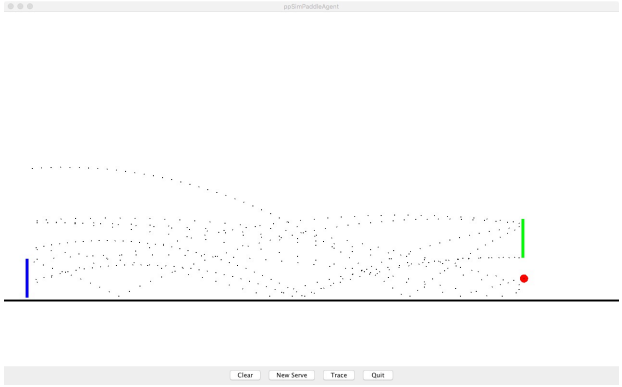

Figure 3

Bonus Questions

4. An obvious extension would be to create a scoreboard object with methods to set the player names, increment and clear the scores of the respective players. The clear method would be called in ppSimPaddleAgent when the Clear button is pressed, the methods which increment the score would be called in ppBall. An example of what this might look like is shown in Figure 4. (15 Bonus Points).

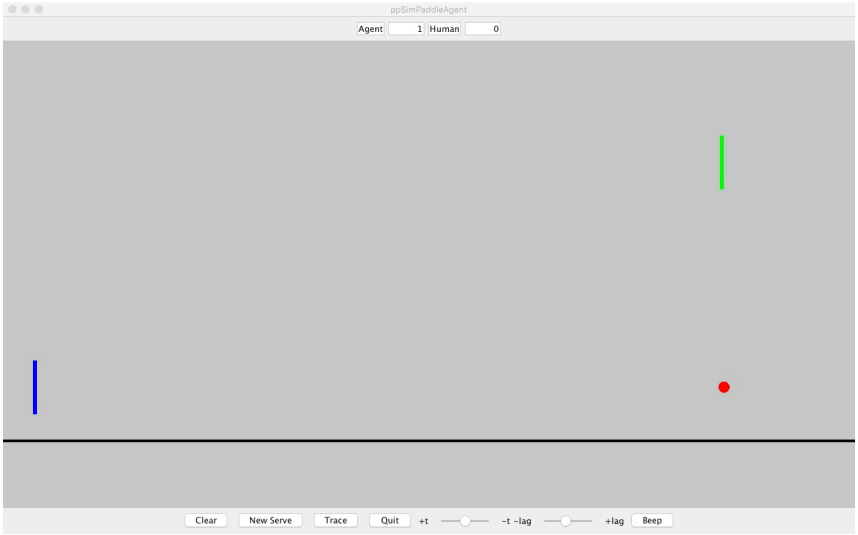

Figure 4

5. The game should also check for a player hitting the ball out of bounds – something that can be done very easily inside of ppBall. Modify the scorekeeping in Part (4) to award a point to the Agent if the Player hits the ball out of bounds and vice-versa. Note: this part will not be counted unless you do Part (4) also. (10 Bonus Points).

With the basic game structure in place, it is not hard to add features. For example the slider labeled +t/-t in Figure 4 slows down play as you slide from left to right; the slider simply changes the multiplier that converts TICK to milliseconds. The slider to its right, labeled –lag/+lag, slows down the reaction time of the agent as it is moved from left to right. This is implemented by updating the position of the paddle every Nth iteration of the while loop in ppPaddleAgent.

To Hand In:

1. The source code files for your project: *ppSimPaddleAgent*, *ppPaddle*, ppPaddleAgent, *ppBall*, *ppTable* and *ppSimParams* and any other classes required to build your application in Eclipse.
2. The output files A4-2.pdf and A4-3.pdf.

Your code must successfully compile and build to be evaluated.

All assignments are to be submitted using myCourses (see myCourses page for ECSE 202 for details).

About Coding Assignments

We encourage students to work together and exchange ideas. However, when it comes to finally sitting down to write your code, this must be done *independently*. Detecting software plagiarism is pretty much automated these days with systems such as MOSS.

https://www.quora.com/How-does-MOSS-Measure-Of-Software-Similarity-Stanford-detect-plagiarism

Please make sure your work is your own. If you are having trouble, the Faculty provides a free tutoring service to help you along. You can also contact the course instructor or the tutor during office hours. There are also numerous online resources – Google is your friend. The point isn't simply to get the assignment out of the way, but to actually learn something in doing.

fpf October 26, 2020
Version 1.0