COP290: Multi Player Ping Pong Game

Aayan Kumar (2014CS10201)
Shreyan Gupta (2014CS10485)
Vaibhav Bhagee (2014CS50297)

6 April, 2016

Contents

1 Introduction

In this assignment we try to implement a 4-player Ping Pong game. This app will to students who are interested in wasting their time.

One or more people, who have nothing else to do in their lives, can play either with an AI or other players over a Wi-Fi/LAN network. A game is visible to all people connected to each other, directly or indirectly, and all such users can play the game.

The game is played as follows: there are 4 paddles. The bottom paddle on the screen is the current player. The current player uses his mouse to move the paddles. There is also a ball moving around. The job of each player is to use the paddle to deflect the ball and prevent it from hitting from his/her wall. In case a player lets the ball hit his/her wall thrice, he/she would lose the game. If there are not enough players to control all 4 paddles, the paddles would be controlled by an AI.

2 Scope of Assignment

2.1 Current Aim

Currently we are focusing on making a basic desktop app which can simulate basic gameplay, with limited GUI as of now, and basic P2P networking.

2.2 Possible future improvements

- The app can be extended to involve gameplay with more than one ball. An array of balls has been declared, whose size can be modified later.
- The app can be made more dynamic by making a ball go off at higher speeds or imparting spin onto the ball to make the trajectories unpredictable and make the game more exciting. This could happen, for example, when the paddle moves at a high velocity while hitting the ball. A mouse click could activate a power-up which makes the ball go at high speeds.
- The 2D interface can be converted into 3D graphics. So, instead of 2D motion, the balls can move around all 3 axes, for example it could bounce up in the air etc.
- We can also include special power-ups to make the game more expansive like high speed ball, gravity well, elongate/ reduce paddle size etc.
- The assignment can be further extended to mobile apps alongside desktop clients for further reach.
- Exhaustive network tests can be constructed and simulated to test the game on High Latency Low Bandwidth networks, packet loss and disconnection of peers

3 Implementation Details

3.1 User Interface

The UI would be implemented as follows:

- The **SWING Library** will be used for rendering objects etc.
- The **MouseClickOrMotion** class has a constructor which initialises all the frames, panels etc. which shall be used to render the board, paddles, balls, obstacles etc. It implements several Listener interfaces which help the various graphic elements respond to mouse events, imparting controllability to the paddles.

- The **Rectangle, Circle** classes are used to maintain rectangles(paddles, board etc) and circles(balls etc) respectively. They also contain some functions which call appropriate drawing functions which help render the shapes on the screen.
- The **BlankArea** class is used to create the whole window on which the game will be rendered.

3.2 Game Engine

The game engine handles the physics of the game which includes the reflection, velocity management, AI (Computer Player), updating position, collision, etc.

- The class **Board** handles the interface between the Networks and UI. It contains appropriate functions to do the same.
- **Interface with Networks** - There are 3 basic functions which interace with the server side to send and receive information about other players
  - **is_pseudo_server()** - This functions returns whether the current player is the pseudo server or not.
  - **broadcast()** - This function is used to broadcast information about the current player to all the other players connected to the network.
  - **get_all_messages()** - This function is used to receive all messages from other players.
- **Interface with UI** - There are 2 basic functions which interact with the UI.
  - **update()** - This function is periodically called at a frequency of about 20Hz to 60Hz. This updates the position of the ball, take care of all state updates, collisions, etc. It returns an Object with would be used to render the position of components and pass information about collisions etc.

– **move_paddle()** - This functions is called to update the position of the paddle of the current$_player$.

- The Aritficial Intelligence component takes care of the motion of the paddles which are not connected to any player.
- The update function renders the position of the components (balls) only when the current player is the **pseudo server**. Otherwise the broadcasted message from the pseudo server is used to update the position of the components.
- The basic laws of reflection would be used to render the motion of the ball. Collision detection (for multiple balls) would use the relative closeness of the coordinates of every pair of balls. Appropriate function would handle these.

3.3 Networking

- **Connection Protocol** The P2P connections will be managed and implemented using UDP (User Datagram Protocol). The use of UDP is justified here as this game as the frequency of exchanging messages (Packets) among the peers is quite high (roughly 40 - 60 times in one second) and hence fast exchange of packets is required. Even if some packets get lost due to Network Latency Issues, the flow of the game will not be hampered much. Further details of handling Network Latency Issues have been discussed below.

- **Basic Message Types** The basic structure of the P2P networking requires messages to be exchanged between all peers on a notionally personal basis rather than a client requesting a server. In our case, there will broadly be 2 classes of messages exchanged -

  - **Network related messages** Related to the connection of new peers to the game, disconnection of existing peers from the game etc
  - **Game State Related Messages** sent by the physics engine to update the ball position, paddle position etc

- **Maintaining Game state across the peers** We have also introduced the concept of a pseudo server - which will be the client with the minimum IP address - which will house the computer player(s). In order to maintain the continuity of the motion of the ball, all the peers will update their paddle positions and the position of the ball(s) returned to them by the physics engine. However, in order to ensure that the same game state is seen by the peers, the peers will exchange and update the paddle positions of their own as well as the other peers, but only the pseudo server will be allowed to share the updated ball(s) position with the other peers through the network.

- **Basic implementation ideas** The clients will have a main RequestHandler class will will provide a port for connecting with the other peers and a run() method. The run method will look for incoming connections and upon recieving one, it spawns a new thread of the ConnectionHandler class and adds the details of the new connection to the hashtable of the former and generates a server message. Upon receiving this message, the other peers will send the request to the IP address of the new peer added and suitably update their hashtables. Finally, to maintain uniformity in the order in which the clients joined the game, the pseudo server will send a message specifying the order of joining and all the other peers will update.

- **Eventflow of messages over the network** The main Thread class which will form the interface between the physics engine and the network threads is the RequestHandler class. This class provides the method broadcast(String message) to broadcast the message to the connected peers. When the physics engine need to broadcast the update game

state to the other peers, it calls the function broadcast() to broadcast the message to other peers. When a connected peer receives a message, the ConnectionHandler thread will check for the type of message - if it is a network related message, then the suitable action will be taken at the network side. If it is a game state related message, then it will be enqueued in the queue, which will be returned to the physics engine when it asks for it for subsequent action.

- **Network Latency and Peer Disconnection** To handle issues of disconnection of a peer, the main RequestHandler thread provides a method ping_all_clients(). This method will ping the peers connected with itself with a dummy packet and wait for the response. If the response does not come within a time period of 1s, then the peer will be assumed to be disconnected. A message will be generated by the ConnectionHandler thread in this regard and sent to the peers. The peers on receiving this message will update the status of the disconnected peer to a computer player and also recompute the pseudo server. This info will be added to the queue to make it available to the physics engine for updation. So now basically the current game will split into 2 games - 1 with 3 human players and a computer player replacing the disconnected person and the other with the disconnected person and 3 computer players.

- **Basic Testing** Basic testing for packet loss and Network latency will be added in the form of packets being accepted with some probability p. Other basic tests include disconnection of a peer during a game.

4 Class Structures

The classes used:

4.1 UI Classes

Listing 1: Class Parameters for Rectangle

```
1
2  public class Rectangle{
3
4      private int midpoint_x;
5      private int midpoint_y;
6      private int length;
7      private int thickness;
8
9      public Rectangle(int midx, int midy, int l, int t)
10     {
11         //Set the values of all the above variables
12     }
```

```
13
14      public void draw(Graphics g)
15      {
16          //Call function to render a rectangle with necessary
17          //parameters
18      }
19
20          //Get and Set Methods for the following variables
21      public int getMidX(){return midpoint_x;}
22
23      public int getMidY(){return midpoint_y;}
24
25      public int getLength(){return length;}
26
27      public int getThickness(){return thickness;}
28
29      public void setMidX(int x){midpoint_x = x;}
30
31      public void setMidY(int y){midpoint_y = y;}
32
33  }
```

Listing 2: Class Parameters for Circle

```
1
2   public class Circle{
3
4       private int center_x;
5       private int center_y;
6       private int diameter;
7
8       public Rectangle(int midx, int midy, int d)
9       {
10          //Set the values of all the above variables
11      }
12
13      public void draw(Graphics g)
14      {
15          //Call function to render a rectangle with necessary
16          //parameters
17      }
18
19          //Get and Set Methods for the following variables
20      public int getCenterX(){return center_x;}
21
22      public int getCenterY(){return center_y;}
```

```
23
24      public int getLength(){return diameter;}
25
26      public void setCenterX(int x){center_x = x;}
27
28      public void setCenterY(int y){center_y = y;}
29
30  }
```

Listing 3: Class Parameters for MouseClickOrMotion

```
1
2   public class MouseClickOrMotion extends JPanel implements
3               MouseMotionListener, MouseListener{
4
5       private BlankArea blankArea = new BlankArea();
6       private int board_side;
7       private int ball_radius;
8       private int[] paddle_length = new int[4];
9       private int[] paddle_height = new int[4];
10      private JTextArea textArea;
11      private final static String newline = "\n";
12      Rectangle p1;
13      Rectangle p2,p3,p4;
14      Rectangle board;
15      Board game_board;
16      private static MouseClickOrMotion newContentPane;
17
18      private static JFrame frame = new JFrame();
19
20      private int min(int a, int b)
21      {
22          //return min of a and b
23      }
24
25      public MouseClickOrMotion() {
26          //Constructor function, initialises all necessary panels,
27          //set sizes, adds event handler to the board panel and sets
28          //default values to the paddles.
29      }
30
31      private void update_coordinates(){
32          //Calls game_board.update() to update positions of all
33          //objects on the board
34      }
35
```

```
36      public void mousePressed(MouseEvent e) {
37         //Event handler
38      }
39
40      public void mouseReleased(MouseEvent e) {
41         //Event handler
42      }
43
44      public void mouseEntered(MouseEvent e) {
45          //Event handler
46      }
47
48      public void mouseExited(MouseEvent e) {
49          //Event handler
50      }
51
52      public void mouseClicked(MouseEvent e) {
53          //Event handler
54      }
55
56      public void mouseMoved(MouseEvent e) {
57          //Event handler, gets the position of the mouse pointer and
58          //uses the x coordinate to update the position of the
59          //current player's paddle
60      }
61
62      public void mouseDragged(MouseEvent e) {
63          //Event handler, gets the position of the mouse pointer and
64          //uses the x coordinate to update the position of the
65          //current player's paddle
66      }
67
68      private static void createAndShowGUI() {
69          //Declares all frames, necessary panels, window event handlers
70      }
71
72      public static void main(String[] args) {
73          //Schedule a job for the event-dispatching thread:
74          //creating and showing this application's GUI.
75          //Also create another thread which calls the
76          //update_coordinates function at 60Hz and updates
77          //the positions of all objects on the board
78      }
79  }
```

Listing 4: Class Parameters for Blank Area

```
1
2  public class BlankArea extends JLabel {
3      Dimension minSize = new Dimension(100, 100);
4      Dimension prefsize = new Dimension(1920,1080);
5      Rectangle r1,r2,r3,r4,r5 = new Rectangle(0,0,0,0);
6      Circle b[]
7
8      public BlankArea(Color color) {
9          //Constructor, set basic background stuff
10     }
11
12     @Override
13     protected void paintComponent(Graphics g) {
14         //Draw all the rectangles
15     }
16
17     public void newRect(Rectangle board,Rectangle bottom, Rectangle
18     left, Rectangle right, Rectangle top) {
19         //Redraw the rec
20     }
21
22     public Dimension getMinimumSize() {
23         //returns minsize
24     }
25
26     @Override
27     public Dimension getPreferredSize() {
28         //returns prefsize
29     }
30 }
```

4.2 Game Engine Classes

Listing 5: Class Parameters for Ball

```
1
2  public class Ball{
3      double posX;     // x coordinate of ball
4      double posY;     // y coordinate of ball
5      double velX;     // x coordinate velocity
6      double velY;     // y coordinate velocity
7      double radius;   // radius of ball
8  }
```

Listing 6: Class Parameters for Paddle

```
public class Paddle{
    int orientation;     // 1 for horizontal and 2 for vertical
    double posX1;        // end point 1
    double posX2;
    double posY1;        // end point 2
    double posY2;
}
```

Listing 7: Class Parameters for Player

```
public class Player{
    String name;
    int player_number;  // defined as 0,1,2,3 Needed for orientation
    boolean is_AI;      // current_player isn't AI
    int level_AI;       // defined the difficulty level of the AI
    int score;          // if required
    int lives;          // number of lives left
    Paddle p;
}
```

Listing 8: Class Parameters for Board

```
public class Board{
    double width;            // width of the board
    double height;           // height of the board
    double energy;           // brief value of the velocity of every ball
    ArrayList<Ball> ball_list;
    Player[] plr = new Player[4];   // player[0] is the current_player

    Object update(){
        // called by UI
        // update the position of the ball
        // take care of reflections
        // return an Object to render the board

        broadcast();
        get_all_messages();
    }
        void move_paddle(){
        // called by UI to update the position of the paddle
    }

    void artificial_intelligence(int i){
        if(plr[i].is_AI){
            // i is the player who's paddle has to be updated
```

```
24          }
25      }
26
27      boolean is_pseudo_server(){
28          // returns true depending on whether the current player is
29          // the pseudo server
30          return RequestHandler.is_pseudo_server();
31      }
32
33
34
35      void broadcast(){
36          if(is_pseudo_server()){
37              // broadcast the position of the ball and all players
38              RequestHandler.broadcast("Appropriate Message");
39          }else{
40              // broadcast the position of the current_player
41              // paddle (Player Object)
42              RequestHandler.broadcast("Appropriate Message");
43          }
44      }
45      void get_all_messages(){
46          PriorityQueue<String> messageQueue = RequestHandler.get_all_messages();
47          // receive the broadcasted message from the server
48          //and decode them appropriately
49          // type of messages
50          // 1) position of ball and all players
51          // 2) position of paddle of a certain player
52          // 3) new player has been added -
53          //    Get me the details of the player along with player number
54      }
55  }
```

4.3 Network Classes

Listing 9: Class Parameters for Request Handler Class

```
1   public class RequestHandler implements Runnable
2   {
3       private HashMap<String,ConnectionObject> peerList;
4       //List of peers connected
5
6       private PriorityQueue<String> messageQueue;
7       //Queue of received messages
8
```

```
 9      private is_pseudo_server;
10      //Returns if the client is a pseudo server or not
11
12      public RequestHandler()
13      {
14          this.peerList = new HashMap<String,ConnectionObject>();
15          this.messageQueue = new PriorityQueue<String>();
16          this.is_pseudo_server = false;
17      }
18
19      /**
20       * Method to handle connections with peers
21       */
22      public void run()
23      {
24          /**
25           * Function to check for connection requests from peers
26           * and start a new thread to handle each connection.
27           * If the person already exists and got disconnected then
28           * restart the thread
29           */
30      }
31
32      public boolean is_pseudo_server()
33      {
34          return this.is_pseudo_server;
35      }
36
37      public void update_is_pseudo_server()
38      {
39          /**
40           * Function updates the pseudo server on connection or
41           * disconnection of a client from the game
42           */
43      }
44
45      public void broadcast(String message)
46      {
47          /**
48           * Broadcast the message to all the connected clients
49           */
50      }
51
52      public void update_player_type()
53      {
54          /**
```

```
55            * Function to update the type (Human/AI) upon disconnection
56            * of a client from the game
57            */
58      }
59
60      public PriorityQueue<String> get_all_messages()
61      {
62          /**
63           * Returns the queue of messages and reinstantiates it
64           */
65      }
66
67      public void ping_all_clients()
68      {
69          /**
70           * Function to ping all the clients periodically
71           * in order to check if someone has disconnected
72           */
73      }
74  }
```

Listing 10: Class Parameters for Connection Handler Class

```
1   public class ConnectionHandler implements Runnable
2   {
3       private String thread_name;
4       // Name of the thread
5
6       private boolean is_sender;
7       // Flag to identify the thread as sender or receiver
8
9       private RequestHandler parent;
10      // Parent master Request Handler thread
11
12      private DatagramSocket socket = null;
13      // Socket to be connected
14
15      private InetAddress ip_addr;
16      // IP address if required
17
18      public ConnectionHandler(String name, boolean is_sender,
19                               RequestHandler parentThread)
20      {
21          this.thread_name = name;
22          this.is_sender = is_sender;
23          this.parent = parentThread;
```

```
24          this.socket = new DatagramSocket();
25      }
26
27      public void run()
28      {
29          /**
30           * Overloaded run method of the thread
31           */
32      }
33
34      public void connect_as_client(String ip_address)
35      {
36          /**
37           * Function to connect to the given ip address
38           * over a specified port through sockets to
39           * interchange data
40           */
41      }
42
43      public void handleMessages()
44      {
45          /**
46           * Based on is_sender this function will behave
47           * in a different manner as to send data and wait
48           * for response or send response to incoming packet
49           */
50      }
51
52      public void sendMessage(String message)
53      {
54          /**
55           * Function to send message to the connected
56           * peer
57           */
58      }
59
60      public void send_dummy_message()
61      {
62          /**
63           * Function to periodically send
64           * dummy messages to the peer to
65           * check for connection losses
66           */
67      }
68  }
```

Listing 11: Class Parameters for Connection Object Class

```
1  public class ConnectionObject
2  {
3      private ConnectionHandler this_thread;
4      // The thread object which handles this connection
5
6      private Date timestamp;
7      // Timestamp of the last received message
8
9      private int player_index;
10     // Index of the player in the game
11
12     public boolean is_human;
13     // Flag to check whether player is human or AI
14 }
```

5 References

- **Tutorial on UDP Socket Programming**
  https://docs.oracle.com/javase/tutorial/networking/datagrams/definition.html
- **Threaded UDP server**
  http://stackoverflow.com/questions/773121/how-can-i-implement-a-threaded-udp-based-server-in-java
- **Rendering a board and event listeners**
  http://www.math.uni-hamburg.de/doc/java/tutorial/uiswing/events/example-1dot4/MouseEventDemo.java
  http://www.math.uni-hamburg.de/doc/java/tutorial/uiswing/events/example-1dot4/MouseMotionEventDemo.java