# 

# 

# 

**Course:** CSE215

**Semester:** Summer 2023

**Section:** 4

**Faculty:** Sfr1

**Project Report**

**Project Name:**

Wireless, intra-network, multi-session, and multiplayer Ping-Pong game.

**GitHub Link:**

https://github.com/SujatAhmed/MultiPlayerPongGame-CSE215finalProject

**Group Number:** 5

**Group Members:**Kazi Sujat Ahmed Sazid (2231177642)  
  
**Link:** <https://github.com/SujatAhmed>  
  
Nishat Tasnim Zaman Nibrita (2132267642)  
  
**Link:** <https://github.com/Nibrita>  
  
Deepeta Zaman Orpa (2222424642)  
  
**Link:** https://github.com/deepetazaman

Jannatul Ferdaus Mim (2311593642)

**Link:** https://github.com/Jannatulferdausmim

# 

# 

# 

# 

# 

# 

# Introduction

A wireless, intra-network, multi-session, and multiplayer Ping-Pong game written in Java.  
  
It is a game where two players can play wirelessly over a local network. It can have multiple server-client configurations and is multi-sessional, meaning any n=2k (k is any integer) amount of players can play on a single server if the server has enough computational power.

# Description

A detailed description of how all the classes work is given below:

## Server

This class starts it all; it has the following UML body:

class Server {

~ Server():

~ sessionInstance: ServerSocket

+ main(String[]): void

}

The server class has a data field called sessionInstance, a server socket. In the constructor, the server socket is instantiated through a specific port. Then, a while loop starts that looks for two new connections each time; once two connections are established, a new thread is launched, a session that will handle the game between these two different connections. In this thread, a loop is set that refreshes 30 times a second. This is our main game loop. In this loop, all the logic-related part of the game is handled, like taking input from both players, updating the ball logic, and giving the players the updated positions of all the components.  
  
In the main method, we just instantiate an instance of the server.

## Client

UML body:

class ClientR {

~ ClientR():

~ socket: Socket

~ WINDOW\_HEIGHT: int

~ br: BufferedReader

~ isr: InputStreamReader

~ osw: OutputStreamWriter

~ data:

String ~ WINDOW\_WIDTH: int

~ bw: BufferedWriter

+ main(String[]): void

}

In the Client class, all the front-end tasks are handled. That is, the client is the player.   
All user interactions are handled, processed, and sent to the server.   
On the flip side, all the outputs from the server are also received here, processed, and then displayed to the player by updating the game panel. All of this is done using the socket object.

## ConnectionHandler

UML body:

class ConnectionHandler {

+ ConnectionHandler():

+ player2: Socket

+ in: InputStreamReader

+ reader: BufferedReader

+ out: OutputStreamWriter

+ player1: Socket

+ writer: BufferedWriter

+ getCoordinatesFromPlayer(Socket): int

+ sendTheUpdatedCoordinates(Socket, int, int, int, int, int, int): void

}

This is the class where all the essential functions for the server to communicate with the clients are defined.  
  
getCoordinatesFromPlayer:

Gets the paddle coordinates from the players.

sendTheUpdatedCoordinates:

Sends the updated paddle coordinates and ball coordinates to both players.

## BallLogic

UML body:  
class BallLogic {

+ BallLogic():

~ ballXVelocity: int

~ ballYVelocity: int

~ by: int

~ XdirectionVector: int

~ random: Random

~ y1: int

~ y2: int

~ bx: int

~ s1: int

~ s2: int

~ YdirectionVector: int

+ runLogic(): String

+ setRandomDirection(): int

+ setCoordinates(int, int, int, int, int, int): String

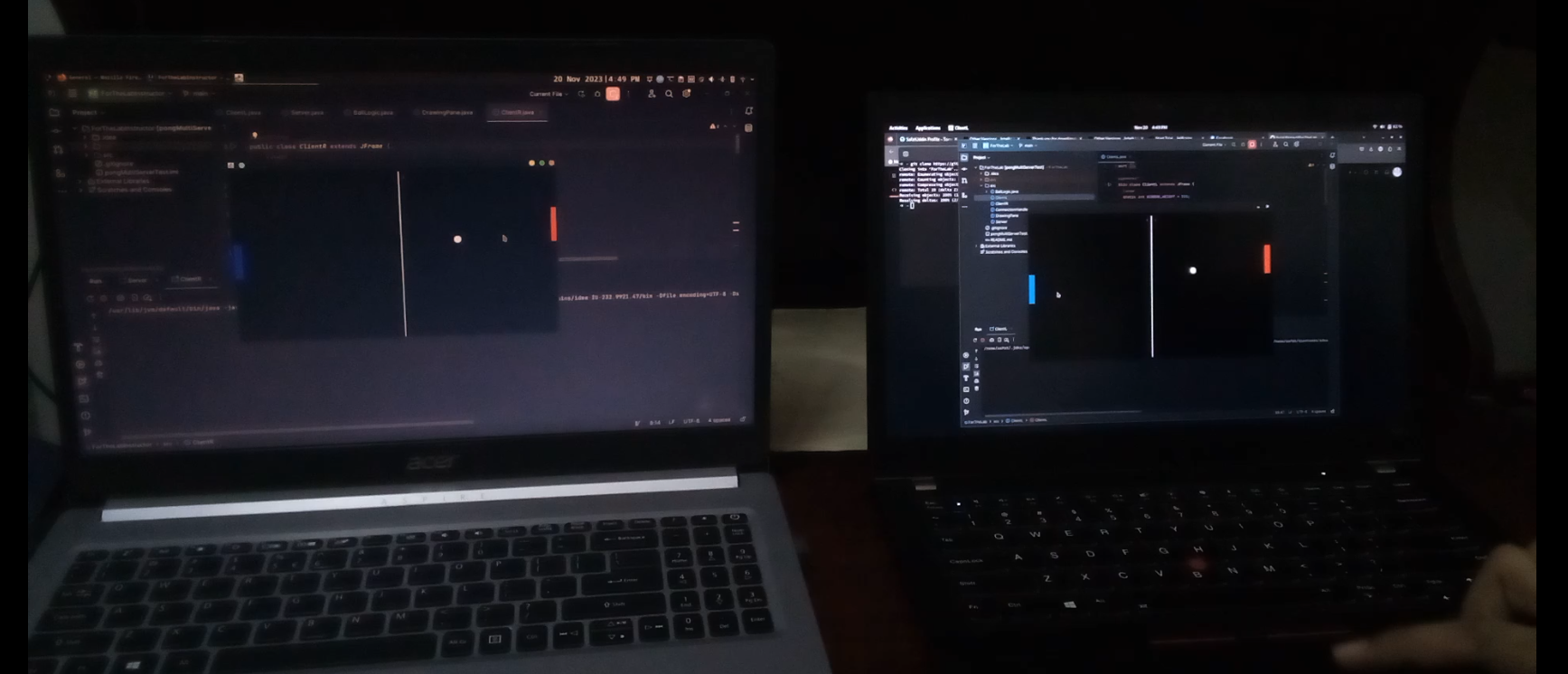
}

This class defines all the logic related to the ball and collision mechanics.

## Configurations

There are multiple possible configurations via which players can play this game.  
Each of them are described below:  
  
**The white PC(ACER) acts as the server in these examples.**

### One PC acts as a server and client, the other as only a client.

In this configuration, one PC acts as the server and the right client, and the other is the left client.

### 

### 

### 

### Single PC only

### 

In this configuration, one PC acts as the server, the right client, and the left client.

  
Players must share a single mousepad to play the game in this mode.

### Multi-Session mode

### 

In this mode, two game sessions can be played with four different players. One PC acts as a server and four client PCs. Unfortunately, we didn’t have this many pcs, so we had to demonstrate using only two. One PC runs the server and handles a session, while the other handles another.  
  
Both sessions are independent of each other and do not affect another session.

# Flowchart

**Start Server -> Start ClientR -> Start ClientL   
The program must be started in this order.**

# 

There are multiple other types of flowcharts, but these two are the most fundamental ones.

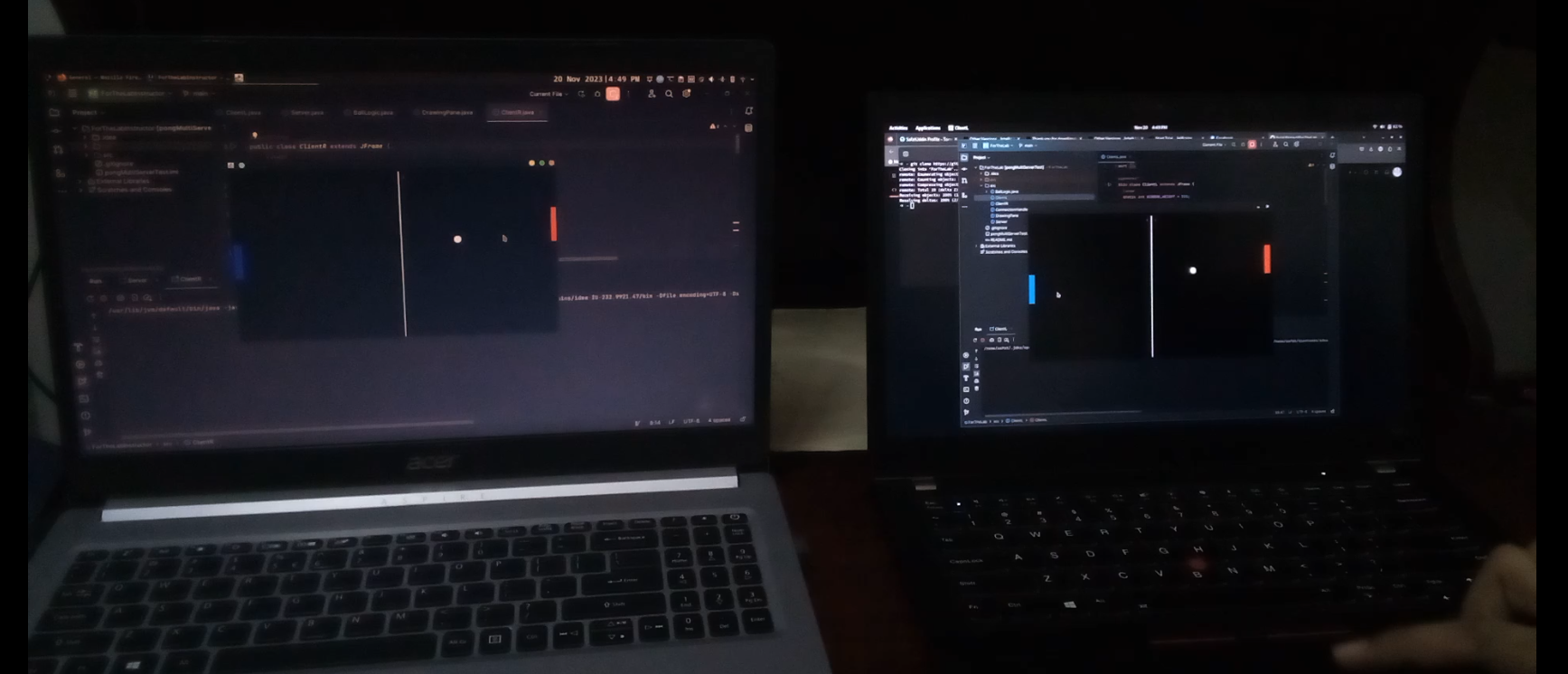
# UML Diagram

### 

## 

## 

# Final Results

  
  
  
**Video Demo Links:**  
<https://photos.app.goo.gl/r6rqs6teLZBEGYmX8>  
**One of the videos here may not be working; in this case, another link is here.**  
<https://drive.google.com/drive/folders/1cRTQSmZ0cPGsb_KIJlwgUK_HbwQAXsPn?usp=sharing>

# Future Work

The intention is to make it inter-network with a server located on the cloud, to which anyone can access and play.

## 

#### 

## 

# 