**PM SHRI KENDRIYA VIDYALAYA KPA RAMAVARMAPURAM, THRISSUR**



Project Final Report

Submitted for the partial fulfillment of CBSE requirement for Computer Science Practicals(083) 2024-25.

**Project Information:**

Project Name : ***GamePort: Multi-Game App***

**Presented By:**

ALBERT JAMES ALOOR

CLASS : XII A

ROLL NO :

**Acknowledgement**

First of all, I would also like to express my gratitude towards my supportive and encouraging computer science teacher, **Smt.** **Latha A Ramakrishnan**, for her inspiring guidance, reliability, constructive criticism and motivational feedback throughout the course of this project.

Finally, l would like to acknowledge my sincere thanks to our respected principal sir, **Shri. B N Talari** for giving me this opportunity to do this investigatory project. I would like to express my sincere gratitude to everyone who supported me throughout this journey, and I will always remain thankful for your contributions.

**Bonafide Certificate**

The undersigned hereby certify that, **ALBERT JAMES ALOOR, Roll no. ,** of **PMSHRI KENDRIYA VIDYALAYA KPA RAMAVARMAPURAM,** session **2024 -25** has completed his project on *'GamePort : Multi-Game App’*.The members have accepted this project as the partial fulfillment of CBSE requirement for Computer Science Practicals(083).

|  |  |
| --- | --- |
| \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_  Internal Examiner  **Latha A Ramakrishnan** | \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_  Principal  **B N Talari** |
| \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_ \_  External Examiner |  |

**DECLARATION**

I, **ALBERT JAMES ALOOR**, hereby declare that this project, titled *'GamePort : Multi-Game App’*, is the record of authentic work carried out by me under the guidance of Smt. Latha A. Ramakrishnan, PGT Computer Science, for the partial fulfillment of the AISSCE 2023-24.

**Signature of the Student**

**Abstract**

This report focuses on the development of a Multi-Game App, GamePort. The project is based on the old multiplayer games containing 3 different games for the players to play. It has a Rock Paper Scissors game in which players can connect locally and play. It contains a space shooter game and Wordler which is an imitation of the infamous Wordle game.

This project explores a new dimension to the traditional games. This game simply aims to bring fun and make you look back to your childhood’s space invaders and other word-guessing riddles on daily newspapers.

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| SL NO | NAME | Page NO |
| 1 | Introduction | 7 |
| 2 | Feasibility Study | 11 |
| 3 | System Design | 13 |
| 4 | Objects Used | 23 |
| 5 | Game Windows | 24 |
| 6 | Source Code | 28 |
| 7 | Game Design | 60 |
| 8 | Conclusion | 65 |
| 8 | Bibliography | 66 |

# Chapter 1 – Introduction

This report describes the process involved in making a Multi-Game App, ‘GamePort’. This chapter discusses the game overview including the synopsis, target audience, description, then focuses on the game design, describing how the game is implemented.

## Game Overview

My App is a Multi-Game App which currently contain Rock Paper Scissors, Space Shooter and Wordler game. The Rock Paper Scissors is a 2d game in which players can connect locally and playe among themselves. The Shooter is an action based shooter game where the main objective of the game is to dodge and shoot the enemy spaceships. Wordler is a game in which the players has to guess a five letter words in 6 tries and win. Pygame is used to look at the designing aspect of the game. Python is the coding language used to give the physics and mechanic concepts. Each aspect of game programming takes a lot of time to master. The objective I hope to deliver by this project is to make a game that has equal amount of designing as well as coding further to strike the perfect balance between the two, So that the game does not feel more physical mundane as well as does not feel jittery in terms of look and feel.

### Synopsis

In the early days, gaming was a solitary adventure. Starting with Tennis for Two and Spacewar! In the 1950s and 60s, and evolving to iconic arcade games like Pong and Pac-Man in the 70s, players were focused on individual challenges and high scores.

With advancing technology, gaming embraced a social dimension. The 80s and 90s introduced local multiplayer, where games like Mario Kart and Street Fighter made gaming a shared experience, full of friendly competition and in-person interactions. Gaming became a social ritual, connecting friends and family face-to-face.

The 2000s brought a shift as online multiplayer enabled players to connect globally. Games like World of Warcraft and Fortnite created vast virtual communities, but often at the cost of the simplicity and closeness of local play. Online lobbies and distant players became the norm, replacing in-person fun with screens.

This project seeks to reignite that original spark. By creating a multi-game app focused on local multiplayer, it brings back the warmth of shared, in-person interactions. Offering a variety of games in a single platform, it allows friends and families to reconnect, enjoying the simple, timeless joy of gaming together in the same room.

### Target Audience

Local multiplayer games were once a trend, which slowly died out with the coming of internet and the human urge for worldwide connectivity. There is no proper media, in which people can locally connect and play various games from a single app. This shows that a large base of players are interested in this genre but is not delivered the proper app which connects all such games and gives me a good opportunity to tap into that market by following the traditions that have already set for this genre while working on implementing all such games in a single app.

But I’m mainly targeting to casual gamer between the age range of 13- 18 and 35+.

### App Description

This App involves two players playing against each other. The Rock Paper Scissors game has a health based approach in which each items thrown by the opponents deduct a specific health from the player (i.e., only when the opponent wins the round) . There is when the health reaches zero, the player loses. The shooter game has also similar health based approach. There is also a limit to the number of bullets a player can fire at a time. In the Wordle game, each player has to guess the correct word in 6 tries.

## Game Objectives

The major objectives of this game project are:

* To create a Multi-game app which contain some traditional multiplayer games.
* To implement health system for Rock Paper Scissors instead of the usual score based system.
* To implement different attack power to rock, paper and scissors.
* To add sound effects to shooting, collisions, etc.
* To design a app in which it is easy to navigate between games and does not require users to put extra time into understanding the functioning of the app.
* To make a User friendly app that is pleasant to look at.
* To code the server for the local multiplayer game.
* To implement a login screen for the players.
* To implement proper file handling to take care of the user’s name password and game statistics.

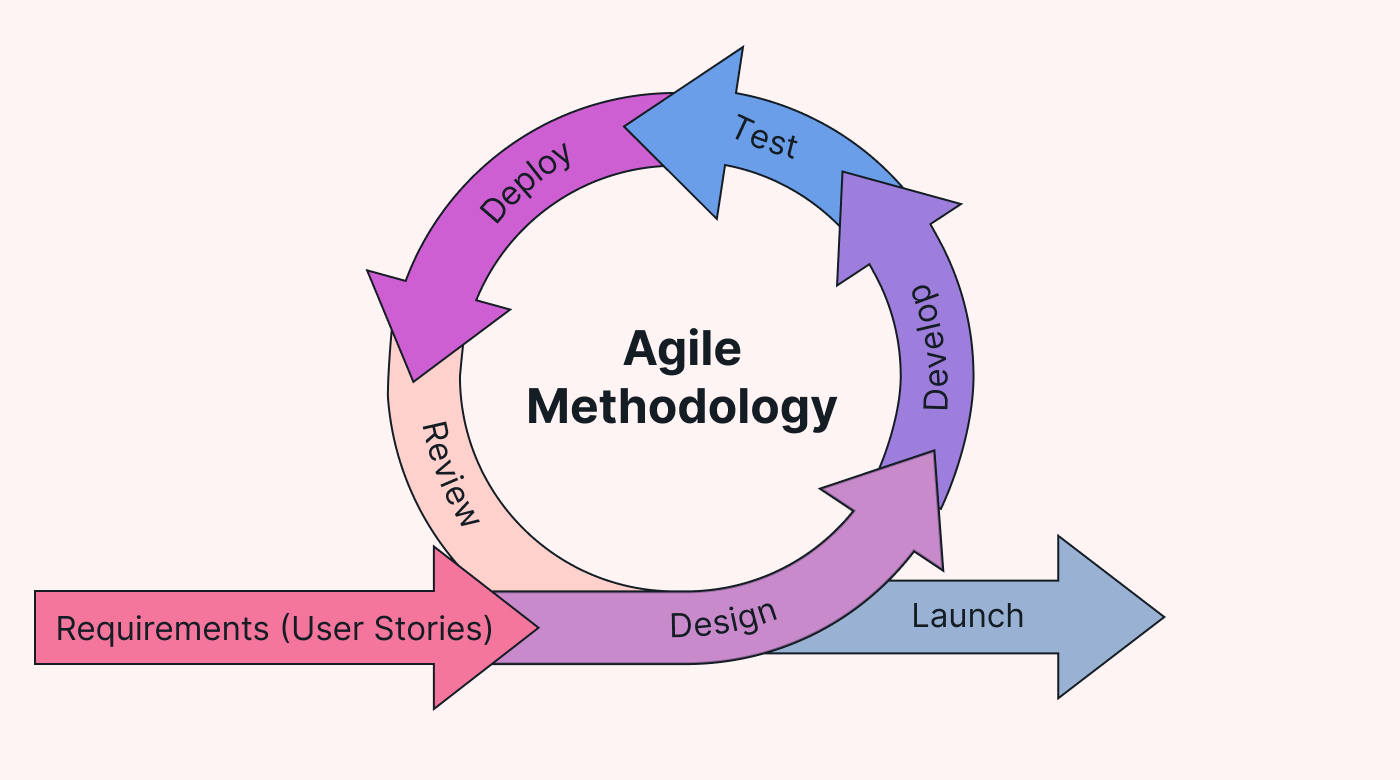
## Why game as a project?

Video games are not just any computer software which are made to benefit user’s daily life, games are rather made for user’s entertainment purpose, so more than anything we need to pay attention to what the user wants from the game, how to make it more entertaining, just making any game will not do, that is why it’s more challenging because I always have to carefully consider if I’m making developing it correctly to entertain users. I also have to invest a lot of time on the proper game designing to make it visually accepted. And to add that game requires a lot of codes. Each part of the code is like blocks of puzzle which has to put together for the proper functioning of the game. Thus I think game is a perfect project to prove myself as a Computer Science student.

## Methodology

The software methodology I have used to develop this game is Agile methodology which is an approach for developing products (in this case: **games**) using short iterations. The main idea is not making the whole project from start to finish, but making small features for the current project in small periods of time. In this way the results of each iteration are used to adjust the project plan.

* Each iteration are small game parts.
* Self-organizing teams generate the most value.
* Regularly reflect and adjust your way of work to boost effectiveness.
* Welcome changing requirements, even late in the project.



# Chapter 2 – Feasibility Study

This chapter describes all the feasibilities that come as questions to both the developers and other users during the development of software. The chapter contains financial feasibility, technical feasibility, economic feasibility and legal feasibility.

## Financial Feasibility

GamePort is a local multiplayer game so there is no hosting cost.

## Technical Feasibility

The tools and technology that were used in the making of the GamePort are:

**Game Library:** Pygame

**Image Editor:** Adobe Photoshop CC 2023 Adobe Illtustrator CC 2019

**CodeEditor:** IDLE Python **Programming Language:** Python

These mentioned above technologies are completely free for students.(except the image editors). There were no other additional tools required to make this game. Most of my time to develop the game was invested in the code editor, IDLE Python.

The simplicity of the project along with the facts mentioned above proves that this software is also technically feasible.

## Economic Feasibility

The resources that are required for this project are:

* + - Development machine. Any regular laptop/PC with a Minimum Ram of 4GB and a decent GPU can be used for the development of this game.
    - Technical tools and software. As mentioned previously, the tools needed to develop this software are available to developers at no charge.

For now I have used free assets to design the game, if I want to release the game for monetizing purposes this project will also require graphic designers for the game to get proper recognition. As these resources are largely free of cost, the only expenses lie in the payment of the machine. Neither of these is scarce in availability nor are they extremely expensive.

## Legal Feasibility

**The Current License Status**

This project is currently unpublished and intended solely for personal use and educational purposes. As such, it does not require a formal license at this time, as it is not distributed or used commercially.

**Planned Approach for distribution**

In the event that the game is prepared for publication or distribution, appropriate legal steps will be taken. This includes selecting a license to govern use and distribution, ensuring compliance with open-source or proprietary guidelines, and reviewing third-party asset licenses.

**Third party assets**

All third-party assets used in the project (e.g., sound effects, images) are documented and sourced from legally compliant resources, such as royalty-free libraries.

**Disclaimer**  
The game is currently in a prototype phase, without any license for distribution or commercial use. Future changes in licensing will be considered and documented as needed.

# Chapter 3– System Design

## List of Modules created and their Purpose:

|  |  |  |
| --- | --- | --- |
| Number | Module Name | Purpose |
| 1 | server.py | Ups the Server |
| 2 | client.py | Runs the main game loop. |
| 3 | game.py | A module containing classes Shooter and Wordler. |
| 4 | multiplayer\_game.py | A module containing RPS class. |
| 5 | button.py | A module containing classes mainButton and Button. |
| 6 | draw\_utils.py | A module containing functions to draw different game windows. |
| 7 | constants.py | A module containing all the constants used in this game. |
| 8 | user\_management.py | A module containing functions to create, update, save and load user’s data. Also has contains the function to log the user in. |
| 9 | sprites.py | A module containing classes Tiles and UIElement. |
| 10 | scan\_server.py | A module containing functions to scan servers that are running in the LAN and connect to it. |
| 11 | network.py | A module containing class Network. |

## List of Other files and their Purpose:

|  |  |  |
| --- | --- | --- |
| Number | Data | Purpose |
| 1 | Assets | A folder containing all the assets used for this game. |
| 2 | words.txt | A text file containing all the five letter words for Wordler. |
| 3 | user\_data | A folder which will potentially contain binary files(user data). |

## List of Classes and their Responsibilities:

|  |  |  |
| --- | --- | --- |
| Number | Class Name | Responsibility |
| 1 | Network | Responsible to connect and handle the connection between the client and the server. |
| 2 | RPS | Responsible to manage the Rock Paper Scissors game, inside the server. |
| 3 | Shooter | Responsible for handling both the spaceships and handle the bullets. |
| 4 | Wordler | Responsible to handle the complete Wordler game |
| 5 | mainButton | Responsible to handle the button on menu screen, and change color on hovering. |
| 6 | Button | A normal button used in the Rock Paper Scissors to get input. |
| 7 | Wordler | Responsible to handle the complete Wordler game |
| 8 | Tile | Is part of UI responsible to animate tiles in wordler. |
| 9 | UIElement | Responsible for drawing notification in wordler |

## List of External modules used and their purpose:

|  |  |  |
| --- | --- | --- |
| Number | Module Name | Purpose |
| 1 | pygame | Is the core of the game. It does everything from providing the UI to playing the music. |
| 2 | socket | To establish a connection between two players |
| 3 | pickle | To read and write binary files used in file handling as well as sending and receiving. |
| 4 | Hashlib | To encode the password of the user. |
| 5 | \_thread | To create multiple threads. |
| 6 | os | to interact with operating system, helps in retrieving user files and assets for the game. |

## Flow charts

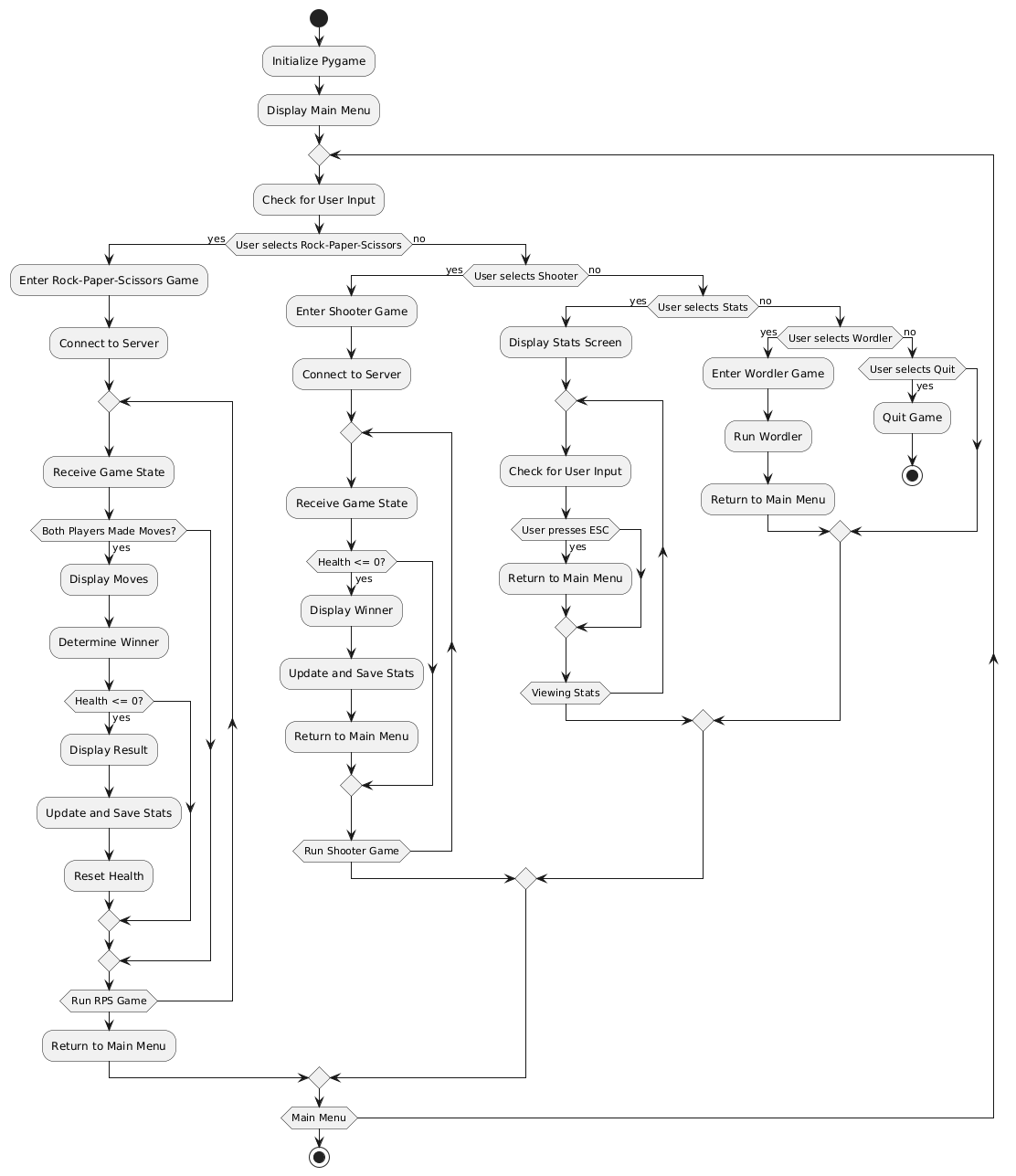
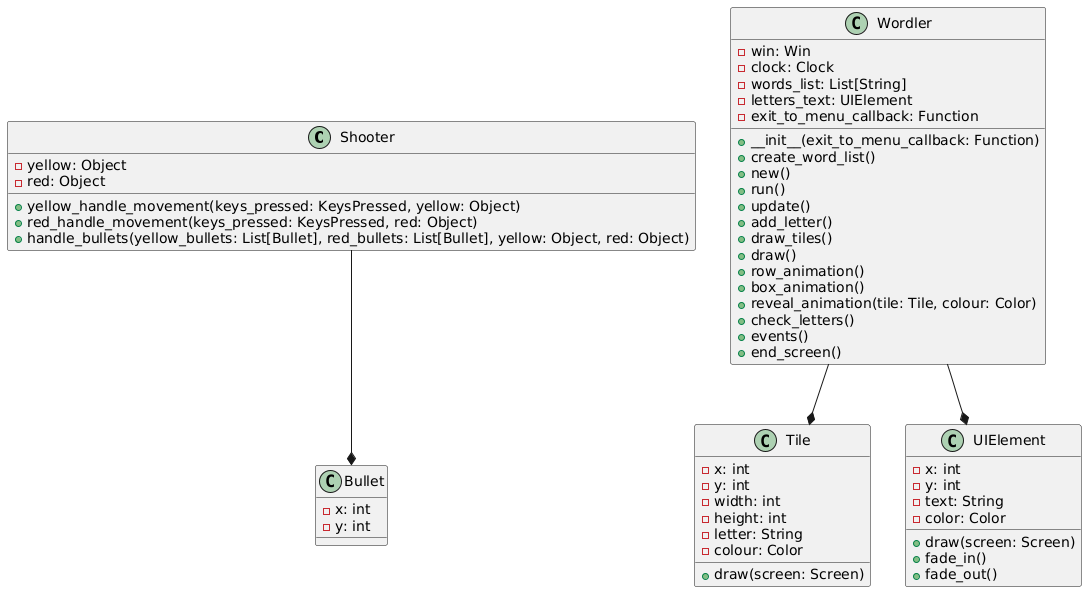
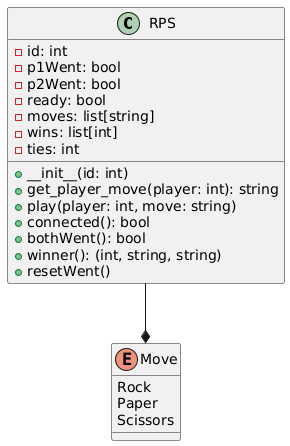
## The following pages contain the flow charts of all modules and class diagrams. These

## diagrams will help you gain a better understanding of the functioning of GamePort.

**server.py**

connect to local host

## server.png

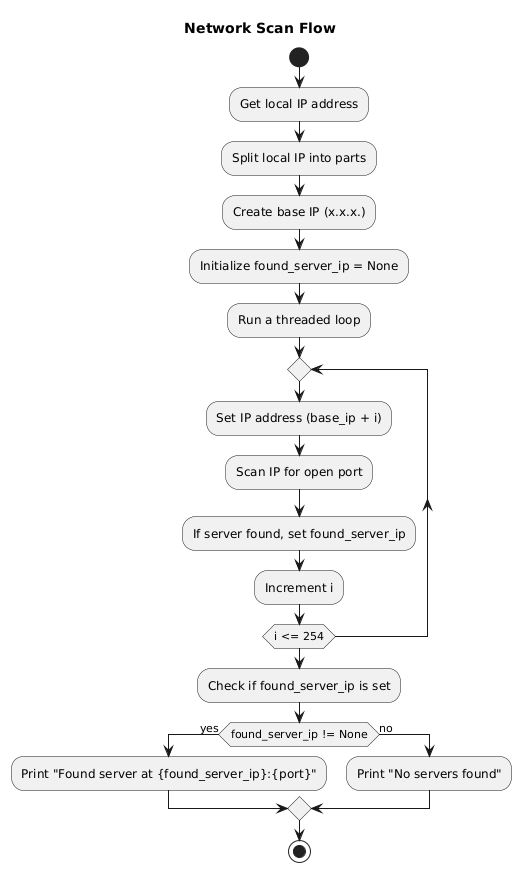
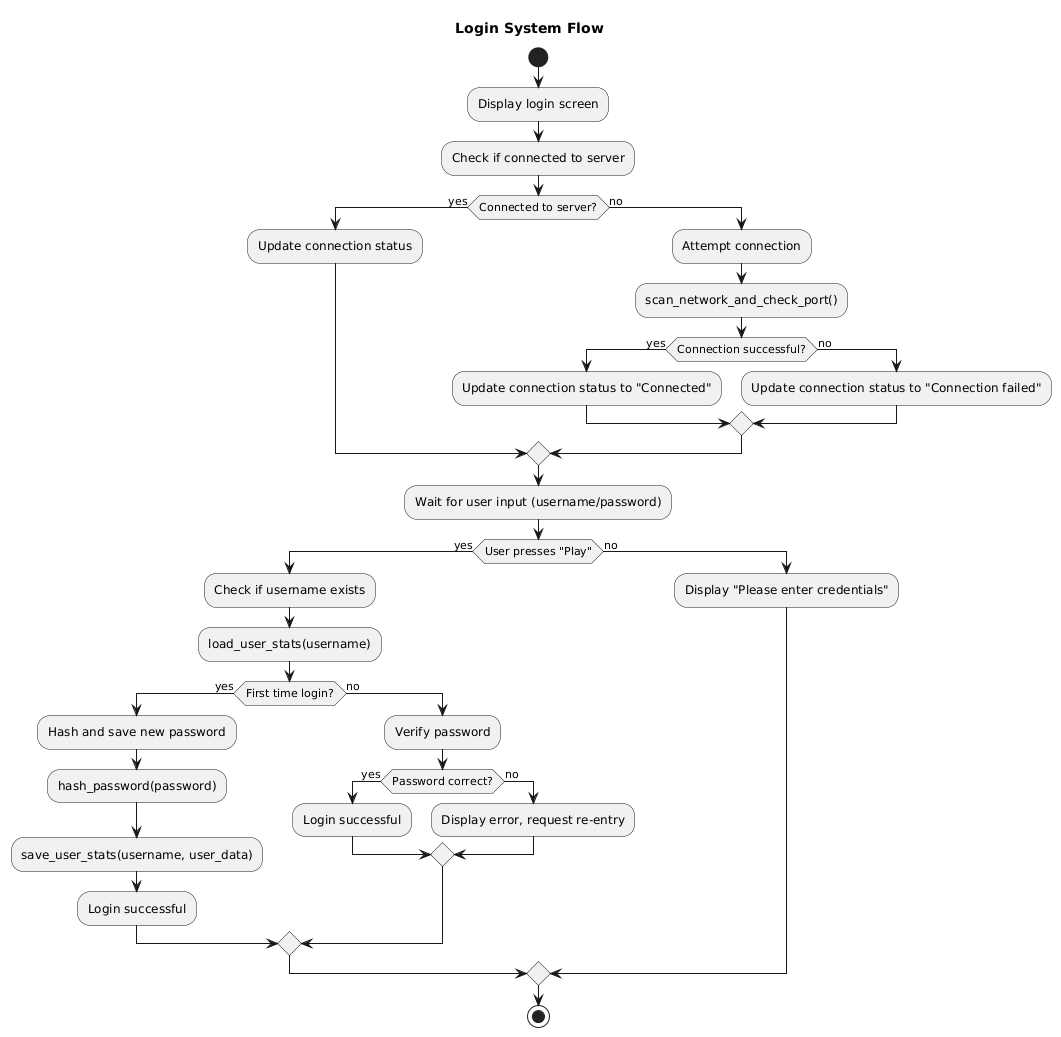
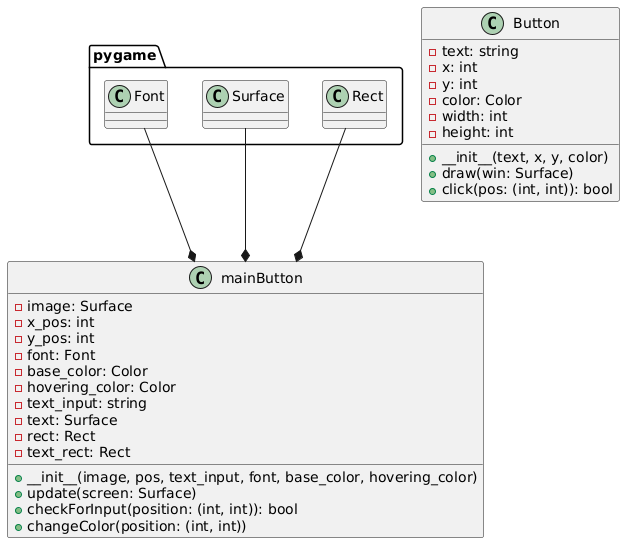
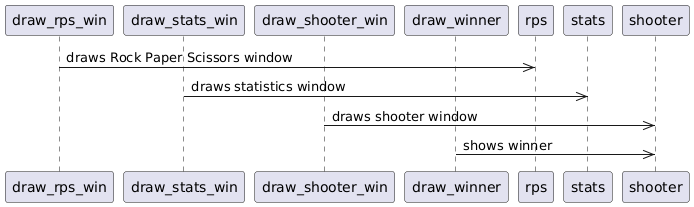
****

**client.py**

**client.py**

**game.py**

**multiplayer\_game.py**

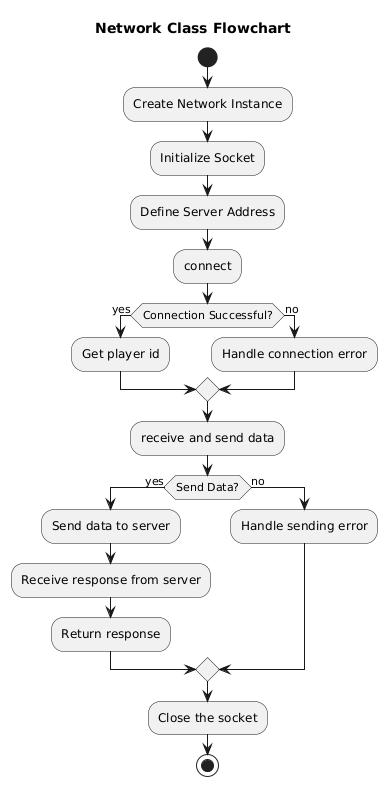


**draw\_utils.py**

**button.py**

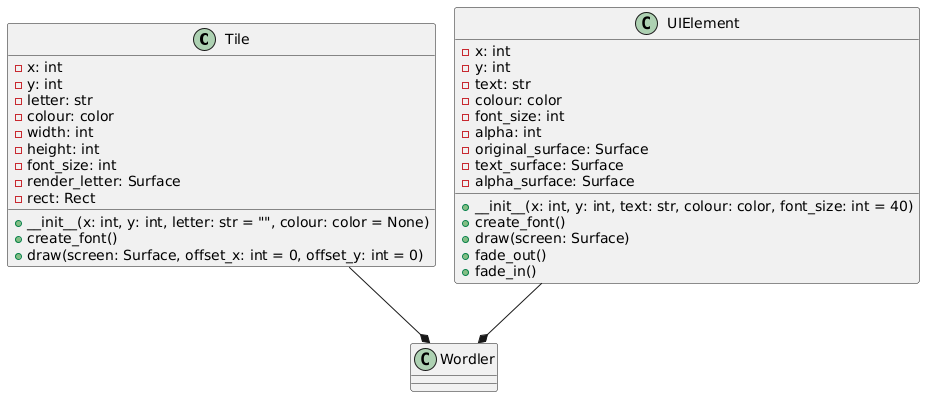
**user\_management.py**

**scar\_server.py**

**

**network.py**

**sprites.py**



# Chapter 4–Obejcts used

This chapter discusses about different objects used in this game, and their capabilities

## Rock Paper Scissors

****

Figure6:Hands

These are the hands used in the Rock Paper Scissors game.

## Shooter

## spaceship_red.pngspaceship_yellow.png

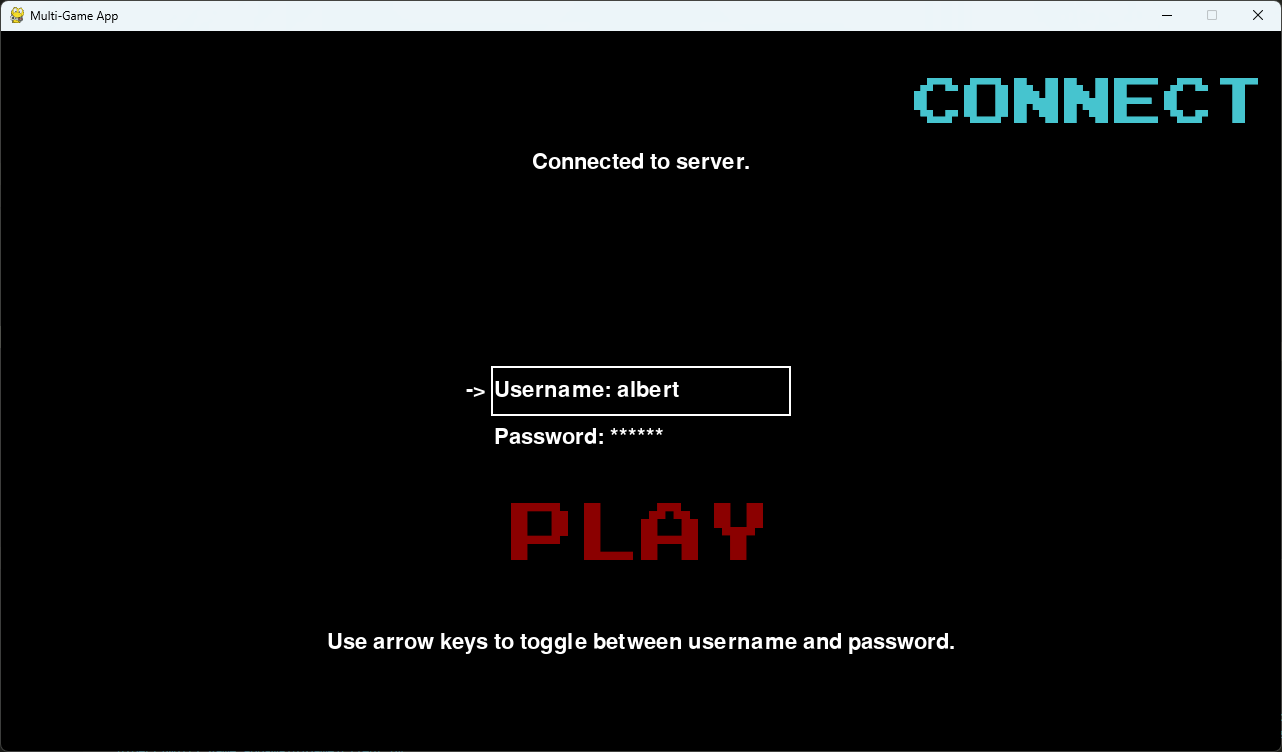
Figure7: Spaceships

These are the space ships used in the Shooter game currently.

There are no other objects used, Wordler game contains some tiles which pygame creates.

This is a simple game and uninterestingly doesn’t have other objects. However I look forward to make my next game project more interactive and fun.

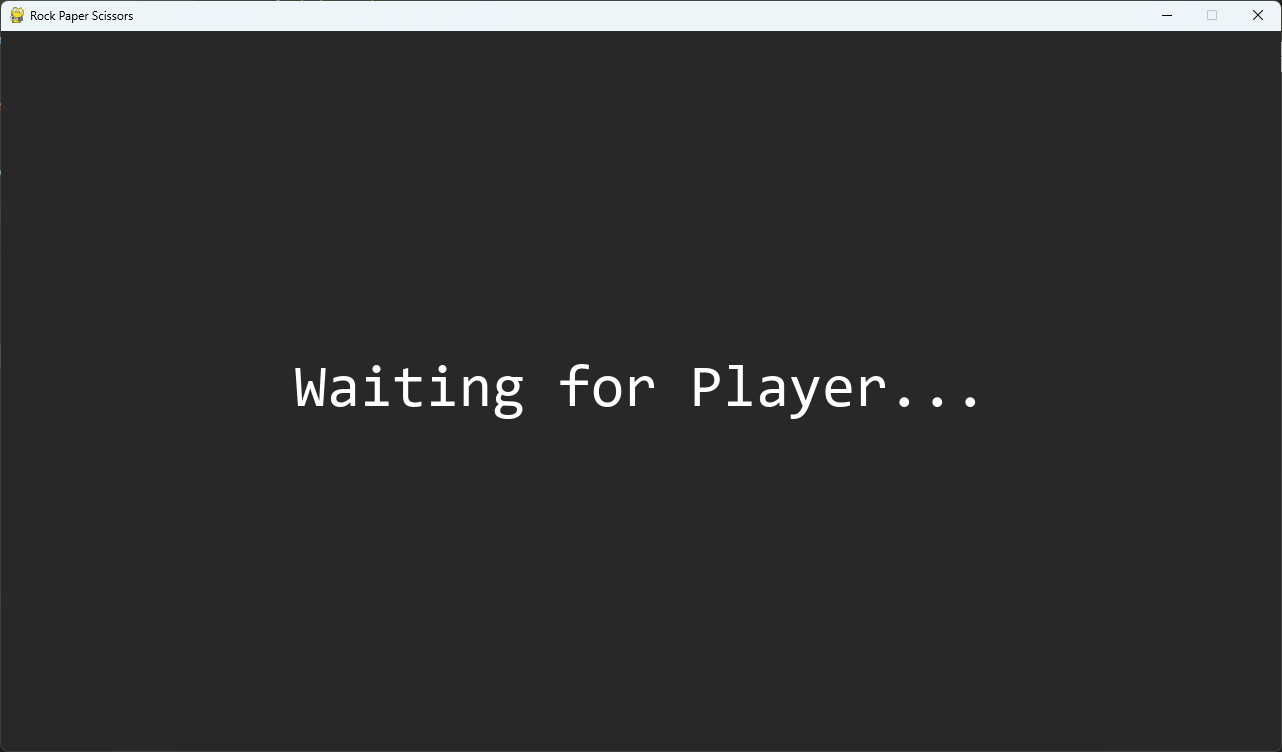
# Chapter 5 – Game Windows



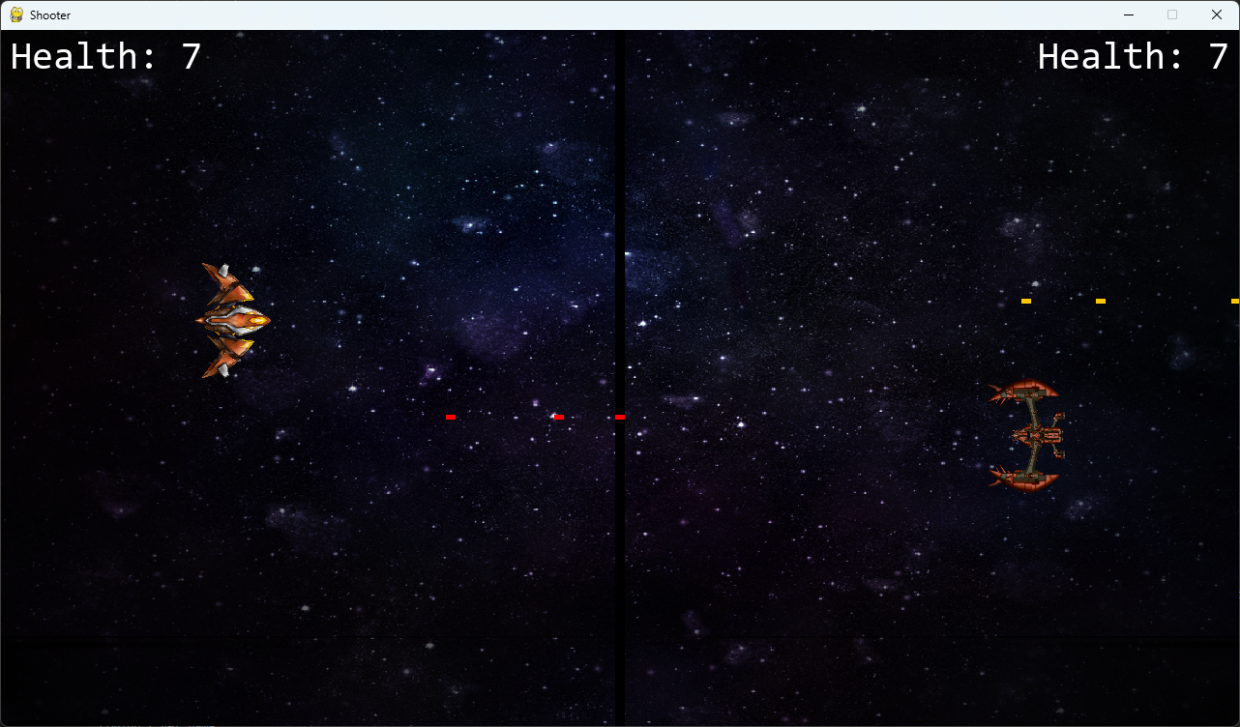
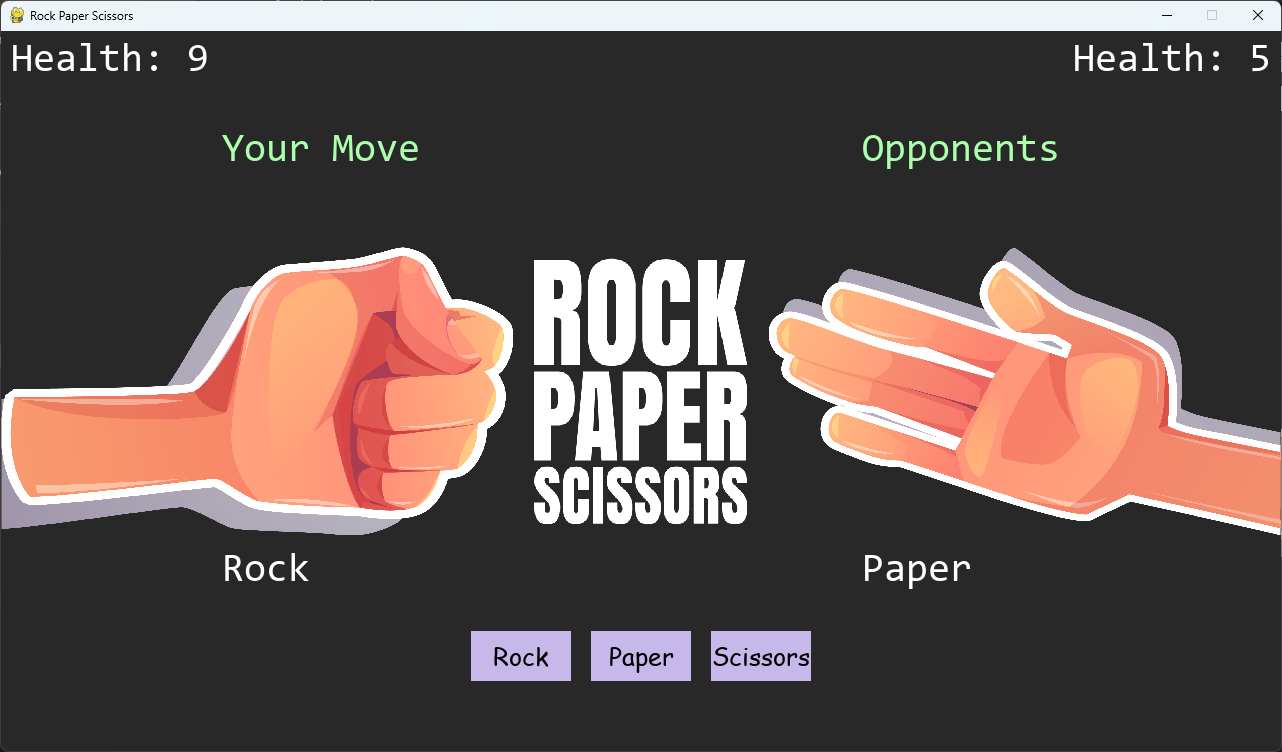
The login window



The main menu



**Rock Paper Scissors waiting window**



**Rock Paper Scissors game window**

**Wordler Game window**

**Shooter Game Window**



**Statistics Window**

# Chapter 6 - Source Code

**server.py**

import socket

from \_thread import \*

import pickle

from multiplayer\_game import RPS

def get\_local\_ip():

    try:

        s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

        s.connect(("8.8.8.8", 80))

        ip = s.getsockname()[0]

    finally:

        s.close()

    return ip

#Get the server's local IP address

server = get\_local\_ip()

# server = "0.0.0.0"

port = 5555

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) #to designate IPv4 protocols

try:

    s.bind((server, port))

except socket.error as e:

    str(e)

s.listen(2)

print("Waiting for a connection, Server Started")

print("Server IP " + server)

connected = set()

games = {}

idCount = 0

def threaded\_client(conn, p, gameId):

    global idCount

    conn.send(str.encode(str(p)))

    reply = ""

    while True:

        try:

            data = conn.recv(4096).decode()

            if gameId in games:

                game = games[gameId]

                if not data:

                    break

                else:

                    if data == "reset":

                        game.resetWent()

                    elif data != "get":

                        game.play(p, data)

                    conn.sendall(pickle.dumps(game))

            else:

                break

        except:

            break

    print("Lost connection")

    try:

        del games[gameId]

        print("Closing Game", gameId)

    except:

        pass

    idCount -= 1

    conn.close()

while True:

    conn, addr = s.accept()

    print("Connected to:", addr)

    idCount += 1

    p = 0

    gameId = (idCount - 1)//2

    if idCount % 2 == 1:

        games[gameId] = RPS(gameId)

        print("Creating a new game...")

    else:

        games[gameId].ready = True

        p = 1

    start\_new\_thread(threaded\_client, (conn, p, gameId))

**Client.py**

import pygame

from network import Network

from button import \*

from constants import \*

from draw\_utils import \*

from game import Shooter, Wordler

from user\_management import \*

pygame.font.init()

pygame.mixer.init()

pygame.mixer.set\_num\_channels(16)

win = pygame.display.set\_mode((width, height))

btns = [Button("Rock", 470, 600,  Pale\_Lavender), Button("Paper", 590, 600,  Pale\_Lavender), Button("Scissors", 710, 600,  Pale\_Lavender)]

#rock paper scissors

def rps(server):

    pygame.display.set\_caption("Rock Paper Scissors")

    run = False

    clock = pygame.time.Clock()

    n = Network(server)

    player\_flag = n.getP()

    font = pygame.font.SysFont("Times New Roman", 150)

    font2 = pygame.font.SysFont("Times New Roman", 60)

    text2 = font2.render("Not connected to server...", 1,  GREEN)

    health\_a = 10

    health\_b = 10

    if player\_flag == None:

        print("Not connected to server")

        win.blit(text2,  (width / 2 - text2.get\_width() / 2, height / 2 - text2.get\_height() / 2))

        pygame.display.update()

        pygame.time.delay(1500)

    else:

        player = int(n.getP())

        print("You are player", player)

        run = True

    while run:

        clock.tick(60)

        try:

            game = n.send("get")

        except:

            run = False

            print("Couldn't get game")

            break

        if game.bothWent():

            draw\_rps\_win(win, game, player, health\_a, health\_b)

            if player == 0:

                if game.get\_player\_move(0) == "Rock":

                    win.blit( PLAYER\_ROCK\_IMAGE, (0, height/2 - height\*0.2))

                elif game.get\_player\_move(0) == "Paper":

                    win.blit( PLAYER\_PAPER\_IMAGE, (0, height/2 - height\*0.2))

                elif game.get\_player\_move(0) == "Scissors":

                    win.blit( PLAYER\_SCISSORS\_IMAGE, (0, height/2 - height\*0.2))

                if game.get\_player\_move(1) == "Rock":

                    win.blit( OPPONENT\_ROCK\_IMAGE, (width\*0.6 , height/2 - height\*0.2))

                elif game.get\_player\_move(1) == "Paper":

                    win.blit( OPPONENT\_PAPER\_IMAGE, (width\*0.6 , height/2 - height\*0.2))

                elif game.get\_player\_move(1) == "Scissors":

                    win.blit( OPPONENT\_SCISSORS\_IMAGE, (width\*0.6 , height/2 - height\*0.2))

            if player == 1:

                if game.get\_player\_move(1) == "Rock":

                    win.blit( PLAYER\_ROCK\_IMAGE, (0, height/2 - height\*0.2))

                elif game.get\_player\_move(1) == "Paper":

                    win.blit( PLAYER\_PAPER\_IMAGE, (0, height/2 - height\*0.2))

                elif game.get\_player\_move(1) == "Scissors":

                    win.blit( PLAYER\_SCISSORS\_IMAGE, (0, height/2 - height\*0.2))

                if game.get\_player\_move(0) == "Rock":

                    win.blit( OPPONENT\_ROCK\_IMAGE, (width\*0.6, height/2 - height\*0.2))

                elif game.get\_player\_move(0) == "Paper":

                    win.blit( OPPONENT\_PAPER\_IMAGE, (width\*0.6 , height/2 - height\*0.2))

                elif game.get\_player\_move(0) == "Scissors":

                    win.blit( OPPONENT\_SCISSORS\_IMAGE, (width\*0.6, height/2 - height\*0.2))

            pygame.time.delay(500)

            try:

                game = n.send("reset")

            except:

                run = False

                print("Couldn't get game")

                break

            winner, p1, p2 = game.winner()

            if (winner == 1 and player == 1):

                if p2 ==  "Rock":

                    health\_b -= 3

                elif p2 ==  "Paper":

                    health\_b -= 2

                else:

                    health\_b -= 1

            elif (winner == 0 and player == 0):

                if p1 ==  "Rock":

                    health\_b -= 3

                elif p1 ==  "Paper":

                    health\_b -= 2

                else:

                    health\_b -= 1

            elif winner == -1:

                pass

            elif (winner == 0 and player == 1):

                if p1 ==  "Rock":

                    health\_a -= 3

                elif p1 ==  "Paper":

                    health\_a -= 2

                else:

                    health\_a -= 1

            elif (winner == 1 and player == 0):

                if p2 ==  "Rock":

                    health\_a -= 3

                elif p2 ==  "Paper":

                    health\_a -= 2

                else:

                    health\_a -= 1

            if health\_a == 0 or health\_b == 0:

                result = ""

                # Show message before resetting the scores

                if health\_b > health\_a:

                    text = font.render("You Won!", 1,  Electric\_Green)

                    result = "win"  # Use = instead of +=

                elif health\_a > health\_b:

                    text = font.render("You Lost...", 1,  Dark\_Red)

                    result = "lose"  # Use = instead of +=

                # Display the result

                win.blit( RECTANGLE, (0, 0))

                win.blit(text, (width / 2 - text.get\_width() / 2, height / 2 - text.get\_height() / 2))

                pygame.display.update()  # Update the display to show the result

                # Update and save user stats

                update\_user\_rps\_stats(stats, result)

                save\_user\_stats(username, stats)

                # Reset scores after the message is shown and stats are updated

                health\_a = health\_b = 0

            pygame.display.update()

            pygame.time.delay(1000)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                run = False

                pygame.quit()

            if event.type == pygame.MOUSEBUTTONDOWN:

                pos = pygame.mouse.get\_pos()

                for btn in btns:

                    if btn.click(pos) and game.connected():

                        if player == 0:

                            if not game.p1Went:

                                n.send(btn.text)

                        else:

                            if not game.p2Went:

                                n.send(btn.text)

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_ESCAPE:

                    health\_a = health\_b = 0

                    n.close()

                    run = False

                    main\_menu(server)

        draw\_rps\_win(win, game, player, health\_a, health\_b)

#shooter game

def shooter(server):

    pygame.display.set\_caption("Shooter")

    red = pygame.Rect(1020, 360,  SPACESHIP\_width,  SPACESHIP\_height)

    yellow = pygame.Rect(200, 360,  SPACESHIP\_width,  SPACESHIP\_height)

    red\_bullets = []

    yellow\_bullets = []

    red\_health = 10

    yellow\_health = 10

    clock = pygame.time.Clock()

    SPACE\_BGM.play(-1)  # Loop the background music indefinitely

    SPACE\_BGM.set\_volume(0.5)  # Lower background music volume

    run = True

    while run:

        clock.tick( FPS)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                run = False

                pygame.quit()

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_LCTRL and len(yellow\_bullets) <  MAX\_BULLETS:

                    bullet = pygame.Rect(yellow.x + yellow.width, yellow.y + yellow.height // 2 - 2, 10, 5)

                    yellow\_bullets.append(bullet)

                    BULLET\_FIRE\_SOUND.play()

                if event.key == pygame.K\_RCTRL and len(red\_bullets) <  MAX\_BULLETS:

                    bullet = pygame.Rect(red.x, red.y + red.height // 2 - 2, 10, 5)

                    red\_bullets.append(bullet)

                    BULLET\_FIRE\_SOUND.play()

                if event.key == pygame.K\_ESCAPE:

                    red\_health = yellow\_health = 10

                    SPACE\_BGM.stop()

                    run = False

                    main\_menu(server)

            if event.type ==  RED\_HIT:

                red\_health -= 1

                BULLET\_HIT\_SOUND.play()

            if event.type ==  YELLOW\_HIT:

                yellow\_health -= 1

                BULLET\_HIT\_SOUND.play()

        winner\_text = ""

        if red\_health <= 0:

            winner\_text = "Yellow Wins!"

        if yellow\_health <= 0:

            winner\_text = "Red Wins!"

        if winner\_text != "":

            WIN\_SOUND.play()

            draw\_winner(winner\_text)

            update\_user\_shooter\_stats(stats, winner\_text)

            save\_user\_stats(username, stats)

            break

        keys\_pressed = pygame.key.get\_pressed()

        Shooter.yellow\_handle\_movement(keys\_pressed, yellow)

        Shooter.red\_handle\_movement(keys\_pressed, red)

        Shooter.handle\_bullets(yellow\_bullets, red\_bullets, yellow, red)

        draw\_shooter\_win(red, yellow, red\_bullets, yellow\_bullets, red\_health, yellow\_health)

    shooter(server)

def stats\_win(server):

    pygame.display.set\_caption("Statistics")

    clock = pygame.time.Clock()

    run = True

    while run:

        clock.tick( FPS)

        stats = load\_user\_stats(username)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                run = False

                pygame.quit()

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_ESCAPE:

                    run = False

                    main\_menu(server)

        draw\_stats\_win(win, stats)

    stats\_win()

def wordler(server):

    game = Wordler(exit\_to\_menu\_callback=lambda: main\_menu(server), victory=lambda: update\_user\_wordler\_stats(stats))

    save\_user\_stats(username, stats)

    run = True

    while run:

        game.new()

        game.run()

def main\_menu(server):

    pygame.display.set\_caption("Multi-Game App")

    run = True

    clock = pygame.time.Clock()

    while run:

        clock.tick(60)

        win.fill(BGCOLOUR)

        MENU\_MOUSE\_POS = pygame.mouse.get\_pos()

        MENU\_TEXT = get\_font(100).render("MAIN MENU", True, "#b68f40")

        MENU\_RECT = MENU\_TEXT.get\_rect(center=(640, 100))

        RPS\_BUTTON = mainButton(image=None, pos=(640, 210),

                            text\_input="Rock Paper Scissors", font=get\_font(50), base\_color= THE\_OTHER\_BLUE, hovering\_color= white)

        SHOOTER\_BUTTON = mainButton(image=None, pos=(640, 320),

                            text\_input="SHOOTER", font=get\_font(50), base\_color= THE\_OTHER\_BLUE, hovering\_color= white)

        WORDLER\_BUTTON = mainButton(image=None, pos=(640, 430),

                            text\_input="WORDLER", font=get\_font(50), base\_color= THE\_OTHER\_BLUE, hovering\_color= white)

        STATS\_BUTTON = mainButton(image=None, pos=(640, 540),

                            text\_input="STATS", font=get\_font(50), base\_color= THE\_OTHER\_BLUE, hovering\_color= white)

        QUIT\_BUTTON = mainButton(image=None, pos=(640, 650),

                            text\_input="QUIT", font=get\_font(70), base\_color= Dark\_Red, hovering\_color= white)

        win.blit(MENU\_TEXT, MENU\_RECT)

        for button in [RPS\_BUTTON, SHOOTER\_BUTTON, WORDLER\_BUTTON, STATS\_BUTTON, QUIT\_BUTTON]:

            button.changeColor(MENU\_MOUSE\_POS)

            button.update(win)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                run = False

            if event.type == pygame.MOUSEBUTTONDOWN:

                if RPS\_BUTTON.checkForInput(MENU\_MOUSE\_POS):

                    rps(server)

                if SHOOTER\_BUTTON.checkForInput(MENU\_MOUSE\_POS):

                    shooter(server)

                if STATS\_BUTTON.checkForInput(MENU\_MOUSE\_POS):

                    stats\_win(server)

                if WORDLER\_BUTTON.checkForInput(MENU\_MOUSE\_POS):

                    wordler(server)

                if QUIT\_BUTTON.checkForInput(MENU\_MOUSE\_POS):

                    pygame.quit()

                    run = False

        pygame.display.update()

while True:

    username, stats, found\_server = login(win)

    main\_menu(found\_server)

**usermanagement.py**

import os

import pickle

import hashlib

import pygame

from constants import USER\_DATA\_DIR, MAX\_USERNAME\_LENGTH, MAX\_PASSWORD\_LENGTH, white, black, width, height, THE\_OTHER\_BLUE, LIGHTGREY, Dark\_Red, get\_font

from button import mainButton

from scan\_server import scan\_network\_and\_check\_port

# Default user statistics structure

default\_stats = {

    "rps\_total\_games": 0,

    "rps\_wins": 0,

    "rps\_losses": 0,

    "rps\_win\_rate": 0.0,

    "shooter\_total\_games": 0,

    "shooter\_red\_wins": 0,

    "shooter\_yellow\_wins": 0,

    "wordler\_guessed\_words": 0,

    "password": ""

}

# Ensure the user data directory exists

if not os.path.exists(USER\_DATA\_DIR):

    os.makedirs(USER\_DATA\_DIR)

# Function to hash password using SHA-256

def hash\_password(password):

    return hashlib.sha256(password.encode()).hexdigest()

# Function to load statistics for a specific user

def load\_user\_stats(username):

    user\_file = os.path.join(USER\_DATA\_DIR, f"{username}.pkl")

    if os.path.exists(user\_file):

        with open(user\_file, 'rb') as f:

            return pickle.load(f)

    return default\_stats.copy()

# Function to save statistics for a specific user

def save\_user\_stats(username, stats):

    user\_file = os.path.join(USER\_DATA\_DIR, f"{username}.pkl")

    with open(user\_file, 'wb') as f:

        pickle.dump(stats, f)

# Function to update user stats based on game result

def update\_user\_rps\_stats(stats, result):

    stats["rps\_total\_games"] += 1

    if result == "win":

        stats["rps\_wins"] += 1

    elif result == "lose":

        stats["rps\_losses"] += 1

    stats["rps\_win\_rate"] = stats["rps\_wins"] / stats["rps\_total\_games"]

def update\_user\_shooter\_stats(stats, result):

    stats["shooter\_total\_games"] += 1

    if result == "Yellow Wins!":

        stats["shooter\_yellow\_wins"] += 1

    elif result == "Red Wins!":

        stats["shooter\_red\_wins"] += 1

def update\_user\_wordler\_stats(stats):

    stats["wordler\_guessed\_words"] += 1

# Function for user login

def login(win):

    pygame.display.set\_caption("Multi-Game App")

    username = ""

    password = ""

    input\_rect = pygame.Rect(width/2- 150, height/2 - 25, 300, 50)

    font = pygame.font.Font(None, 32)

    active = True

    is\_username = True  # Toggle between username and password input

    login\_successful = False

    user\_data = None

    found\_server = ""

    connection\_status = "Please connect to the server."

    while not login\_successful:

        win.fill(black)

        text\_surface = font.render(f"Username: {username}", True, white)

        password\_surface = font.render(f"Password: {'\*' \* len(password)}", True, white)

        notice\_surface = font.render("Use arrow keys to toggle between username and password.", True, white)

        connection\_surface = font.render(connection\_status, True, white)

        understand\_surface = font.render("->",  True, white)

        MOUSE\_POS = pygame.mouse.get\_pos()

        CONNECT\_BUTTON = mainButton(image=None, pos=(width \* 0.85, height \* 0.1),

                            text\_input="CONNECT", font=get\_font(50), base\_color=THE\_OTHER\_BLUE, hovering\_color=LIGHTGREY)

        PLAY\_BUTTON = mainButton(image=None, pos=(width / 2, height \* 0.7),

                            text\_input="PLAY", font=get\_font(65), base\_color=Dark\_Red, hovering\_color=white)

        win.blit(text\_surface, (input\_rect.x + 3, input\_rect.y + 13))

        win.blit(password\_surface, (input\_rect.x + 3, input\_rect.y + 60))

        win.blit(notice\_surface, (width/2 - notice\_surface.get\_width()/2, 600))

        win.blit(connection\_surface, (width/2 - connection\_surface.get\_width()/2, 120))

        if is\_username:

           win.blit(understand\_surface, (input\_rect.x - 25, input\_rect.y + 13))

        else:

            win.blit(understand\_surface, (input\_rect.x - 25, input\_rect.y + 60))

        for button in [PLAY\_BUTTON, CONNECT\_BUTTON]:

            button.changeColor(MOUSE\_POS)

            button.update(win)

        pygame.draw.rect(win, white, input\_rect, 2)

        for event in pygame.event.get():

            if "Connecting" in connection\_status:

                # Call the server connection function if it hasn’t connected

                found\_server = scan\_network\_and\_check\_port(5555)

                if found\_server:

                    connection\_status = "Connected to server."

                else:

                    connection\_status = "Connection failed. Please retry."

                pygame.display.update()

                pygame.time.delay(50)

            if event.type == pygame.QUIT:

                pygame.quit()

                exit()

            if event.type == pygame.MOUSEBUTTONDOWN:

                if CONNECT\_BUTTON.checkForInput(MOUSE\_POS):

                    connection\_status = "Connecting..."

                elif PLAY\_BUTTON.checkForInput(MOUSE\_POS):

                    if is\_username:

                        if len(username) == 0:

                            username = "player"

                        user\_data = load\_user\_stats(username)

                        if user\_data["password"] == "":

                            # First time login, set password

                            user\_data["password"] = hash\_password(password)

                            save\_user\_stats(username, user\_data)

                            login\_successful = True

                        elif user\_data["password"] == hash\_password(password):

                            login\_successful = True

                    else:

                        is\_username = not is\_username  # Switch back to username input after enter

            if event.type == pygame.KEYDOWN:

                if active:

                    if event.key == pygame.K\_BACKSPACE:

                        if is\_username and len(username) > 0:

                            username = username[:-1]

                        elif not is\_username and len(password) > 0:

                            password = password[:-1]

                    elif event.key == pygame.K\_UP:

                        is\_username = not is\_username

                    elif event.key == pygame.K\_DOWN:

                        is\_username = not is\_username

                    elif event.key == pygame.K\_RETURN:

                        if is\_username:

                            if len(username) == 0:

                                username = "player"

                            user\_data = load\_user\_stats(username)

                            if user\_data["password"] == "":

                                # First time login, set password

                                user\_data["password"] = hash\_password(password)

                                save\_user\_stats(username, user\_data)

                                login\_successful = True

                            elif user\_data["password"] == hash\_password(password):

                                login\_successful = True

                        else:

                            is\_username = not is\_username  # Switch back to username input after enter

                    else:

                        if is\_username and len(username) < MAX\_USERNAME\_LENGTH:

                            username += event.unicode

                        elif not is\_username and len(password) < MAX\_PASSWORD\_LENGTH:

                            password += event.unicode

        pygame.display.update()

    return username, user\_data, found\_server

**network.py**

import socket

import pickle

class Network:

    def \_\_init\_\_(self, found\_server):

        self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

        self.server =  found\_server

        self.port = 5555

        self.addr = (self.server, self.port)

        self.p = self.connect()

    def getP(self):

        return self.p

    def connect(self):

        try:

            self.client.connect(self.addr)

            return self.client.recv(2048).decode()

        except:

            pass

    def send(self, data):

        try:

            self.client.send(str.encode(data))

            return pickle.loads(self.client.recv(2048\*2))

        except socket.error as e:

            print(e)

    def close(self):

        self.client.close()

**multiplayer\_game.py**

class RPS:

    def \_\_init\_\_(self, id):

        self.p1Went = False

        self.p2Went = False

        self.ready = False

        self.id = id

        self.moves = [None, None]

        self.wins = [0,0]

        self.ties = 0

    def get\_player\_move(self, p):

        """

        :param p: [0,1]

        :return: Move

        """

        return self.moves[p]

    def play(self, player, move):

        self.moves[player] = move

        if player == 0:

            self.p1Went = True

        else:

            self.p2Went = True

    def connected(self):

        return self.ready

    def bothWent(self):

        return self.p1Went and self.p2Went

    def winner(self):

        p1 = self.moves[0]

        p2 = self.moves[1]

        RPS = ["Rock", "Paper", "Scissors"]

        winner = -1

        if RPS.index(p1) - RPS.index(p2) in [-1, 2]:

            winner = 1

        if RPS.index(p1) - RPS.index(p2) in [1, -2]:

            winner = 0

        return winner, p1, p2

    def resetWent(self):

        self.p1Went = False

        self.p2Went = False

**game.py**

import pygame

import random

from constants import \*

from sprites import \*

class Shooter:

    def yellow\_handle\_movement(keys\_pressed, yellow):

        if keys\_pressed[pygame.K\_a] and yellow.x -  VEL > 0:  # LEFT

            yellow.x -=  VEL

        if keys\_pressed[pygame.K\_d] and yellow.x +  VEL + yellow.width <  BORDER.x:  # RIGHT

            yellow.x +=  VEL

        if keys\_pressed[pygame.K\_w] and yellow.y -  VEL > 0:  # UP

            yellow.y -=  VEL

        if keys\_pressed[pygame.K\_s] and yellow.y +  VEL + yellow.height < height - 10:  # DOWN

            yellow.y +=  VEL

    def red\_handle\_movement(keys\_pressed, red):

        if keys\_pressed[pygame.K\_LEFT] and red.x -  VEL >  BORDER.x:  # LEFT

            red.x -=  VEL

        if keys\_pressed[pygame.K\_RIGHT] and red.x +  VEL + red.width < width:  # RIGHT

            red.x +=  VEL

        if keys\_pressed[pygame.K\_UP] and red.y -  VEL > 0:  # UP

            red.y -=  VEL

        if keys\_pressed[pygame.K\_DOWN] and red.y +  VEL + red.height < height - 10:  # DOWN

            red.y +=  VEL

    def handle\_bullets(yellow\_bullets, red\_bullets, yellow, red):

        for bullet in yellow\_bullets:

            bullet.x +=  BULLET\_VEL

            if red.colliderect(bullet):

                pygame.event.post(pygame.event.Event( RED\_HIT))

                yellow\_bullets.remove(bullet)

            elif bullet.x > width:

                yellow\_bullets.remove(bullet)

        for bullet in red\_bullets:

            bullet.x -=  BULLET\_VEL

            if yellow.colliderect(bullet):

                pygame.event.post(pygame.event.Event( YELLOW\_HIT))

                red\_bullets.remove(bullet)

            elif bullet.x < 0:

                red\_bullets.remove(bullet)

class Wordler:

    def \_\_init\_\_(self, exit\_to\_menu\_callback, victory):

        pygame.init()

        self.win = pygame.display.set\_mode((width, height))

        pygame.display.set\_caption("Wordler")

        self.clock = pygame.time.Clock()

        self.create\_word\_list()

        self.letters\_text = UIElement(width \* 0.33, height \* 0.05, "Not Enough Letters", white)

        self.exit\_to\_menu = exit\_to\_menu\_callback

        self.victory = victory

    def create\_word\_list(self):

        with open("words.txt", "r") as file:

            self.words\_list = file.read().splitlines()

    def new(self):

        self.word = random.choice(self.words\_list).upper()

        print(self.word)

        self.text = ""

        self.current\_row = 0

        self.tiles = []

        self.create\_tiles()

        self.flip = True

        self.not\_enough\_letters = False

        self.timer = 0

    def create\_tiles(self):

        for row in range(6):

            self.tiles.append([])

            for col in range(5):

                self.tiles[row].append(Tile((col \* (TILESIZE + GAPSIZE)) + MARGIN\_X, (row \* (TILESIZE + GAPSIZE)) + MARGIN\_Y))

    def run(self):

        self.playing = True

        while self.playing:

            self.clock.tick(FPS)

            self.events()

            self.update()

            self.draw()

    def update(self):

        self.add\_letter()

    def add\_letter(self):

        # empty all the letter in the current row

        for tile in self.tiles[self.current\_row]:

            tile.letter = ""

        # add the letters typed to the current row

        for i, letter in enumerate(self.text):

            self.tiles[self.current\_row][i].letter = letter

            self.tiles[self.current\_row][i].create\_font()

    def draw\_tiles(self):

        for row in self.tiles:

            for tile in row:

                tile.draw(self.win)

    def draw(self):

        self.win.fill(BGCOLOUR)

        # display the not enough letters text

        if self.not\_enough\_letters:

            self.timer += 1

            self.letters\_text.fade\_in()

            if self.timer > 90:

                self.not\_enough\_letters = False

                self.timer = 0

        else:

            self.letters\_text.fade\_out()

        self.letters\_text.draw(self.win)

        self.draw\_tiles()

        pygame.display.flip()

    def row\_animation(self):

        # row shaking if not enough letters is inputted

        self.not\_enough\_letters = True

        start\_pos = self.tiles[0][0].x

        amount\_move = 4

        move = 3

        win\_copy = self.win.copy()

        win\_copy.fill(BGCOLOUR)

        for row in self.tiles:

            for tile in row:

                if row != self.tiles[self.current\_row]:

                    tile.draw(win\_copy)

        while True:

            while self.tiles[self.current\_row][0].x < start\_pos + amount\_move:

                self.win.blit(win\_copy, (0, 0))

                for tile in self.tiles[self.current\_row]:

                    tile.x += move

                    tile.draw(self.win)

                self.clock.tick(FPS)

                pygame.display.flip()

            while self.tiles[self.current\_row][0].x > start\_pos - amount\_move:

                self.win.blit(win\_copy, (0, 0))

                for tile in self.tiles[self.current\_row]:

                    tile.x -= move

                    tile.draw(self.win)

                self.clock.tick(FPS)

                pygame.display.flip()

            amount\_move -= 2

            if amount\_move < 0:

                break

    def box\_animation(self):

        # tile scale animation for every letter inserted

        for tile in self.tiles[self.current\_row]:

            if tile.letter == "":

                win\_copy = self.win.copy()

                for start, end, step in ((0, 6, 1), (0, -6, -1)):

                    for size in range(start, end, 2\*step):

                        self.win.blit(win\_copy, (0, 0))

                        tile.x -= size

                        tile.y -= size

                        tile.width += size \* 2

                        tile.height += size \* 2

                        surface = pygame.Surface((tile.width, tile.height))

                        surface.fill(BGCOLOUR)

                        self.win.blit(surface, (tile.x, tile.y))

                        tile.draw(self.win)

                        pygame.display.flip()

                        self.clock.tick(FPS)

                    self.add\_letter()

                break

    def reveal\_animation(self, tile, colour):

        # reveal colours animation when user input the whole word

        win\_copy = self.win.copy()

        while True:

            surface = pygame.Surface((tile.width + 5, tile.height + 5))

            surface.fill(BGCOLOUR)

            win\_copy.blit(surface, (tile.x, tile.y))

            self.win.blit(win\_copy, (0, 0))

            if self.flip:

                tile.y += 6

                tile.height -= 12

                tile.font\_y += 4

                tile.font\_height = max(tile.font\_height - 8, 0)

            else:

                tile.colour = colour

                tile.y -= 6

                tile.height += 12

                tile.font\_y -= 4

                tile.font\_height = min(tile.font\_height + 8, tile.font\_size)

            if tile.font\_height == 0:

                self.flip = False

            tile.draw(self.win)

            pygame.display.update()

            self.clock.tick(FPS)

            if tile.font\_height == tile.font\_size:

                self.flip = True

                break

    def check\_letters(self):

        # algorithm to check if the letters inputted correspond to any of the letters in the actual word

        copy\_word = [x for x in self.word]

        for i, user\_letter in enumerate(self.text):

            colour = LIGHTGREY

            for j, letter in enumerate(copy\_word):

                if user\_letter == letter:

                    colour = YELLOW

                    if i == j:

                        colour = GREEN

                    copy\_word[j] = ""

                    break

            # reveal animation

            self.reveal\_animation(self.tiles[self.current\_row][i], colour)

    def events(self):

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                pygame.quit()

                quit(0)

            if event.type == pygame.KEYDOWN:

                if event.key == pygame.K\_ESCAPE:

                    self.playing = False  # Exit the game loop

                    self.exit\_to\_menu()

                if event.key == pygame.K\_RETURN:

                    if len(self.text) == 5:

                        # check all letters

                        self.check\_letters()

                        # if the text is correct or the player has used all his turns

                        if self.text == self.word or self.current\_row + 1 == 6:

                            # player lose, lose message is sent

                            if self.text != self.word:

                                self.end\_win\_text = UIElement(width \* 0.33, height \* 0.9, f"THE WORD WAS: {self.word}", white)

                            # player win, send win message

                            else:

                                self.end\_win\_text = UIElement(width \* 0.36, height \* 0.9, "YOU GUESSED RIGHT", white)

                                self.victory()

                            # restart the game

                            self.playing = False

                            self.end\_win()

                            break

                        self.current\_row += 1

                        self.text = ""

                    else:

                        # row animation, not enough letters message

                        self.row\_animation()

                elif event.key == pygame.K\_BACKSPACE:

                    self.text = self.text[:-1]

                else:

                    if len(self.text) < 5 and event.unicode.isalpha():

                        self.text += event.unicode.upper()

                        self.box\_animation()

    def end\_win(self):

        play\_again = UIElement(85, 750, "PRESS ENTER TO PLAY AGAIN", white, 30)

        while True:

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    pygame.quit()

                    quit(0)

                if event.type == pygame.KEYDOWN:

                    if event.key == pygame.K\_RETURN:

                        return

            self.win.fill(BGCOLOUR)

            self.draw\_tiles()

            self.end\_win\_text.fade\_in()

            self.end\_win\_text.draw(self.win)

            play\_again.fade\_in()

            play\_again.draw(self.win)

            pygame.display.flip()

**draw\_utils.py**

import pygame

from constants import \*

from user\_management import load\_user\_stats

from button import Button

from random import randrange as random\_bluff

btns = [Button("Rock", 470, 600,  Pale\_Lavender), Button("Paper", 590, 600,  Pale\_Lavender), Button("Scissors", 710, 600,  Pale\_Lavender)]

def draw\_rps\_win(win, game, p, health\_a, health\_b):

    win.fill( BGCOLOUR)

    if not(game.connected()):

        font = pygame.font.SysFont("Consolas", 60)

        text = font.render("Waiting for Player...", 1,  white)

        win.blit(text, (width/2 - text.get\_width()/2, height/2 - text.get\_height()/2))

    else:

        win.blit( BACKGROUND\_TITLE,(width/2 -  BACKGROUND\_TITLE.get\_width()/2, height/2 -  BACKGROUND\_TITLE.get\_height()/2))

        font = pygame.font.SysFont("Consolas", 40)

        text = font.render("Your Move", 1,  Mint\_Green)

        win.blit(text, (width/4 - text.get\_width()/2, 100))

        text = font.render("Opponents", 1,  Mint\_Green)

        win.blit(text, (width\*0.75 - text.get\_width()/2, 100))

        Player0\_Health = font.render(f"Health: {health\_a}", 1,  white)

        Player1\_Health = font.render(f"Health: {health\_b}", 1,  white)

        win.blit(Player0\_Health, (10, 10))

        win.blit(Player1\_Health, (width - Player0\_Health.get\_width() - 10, 10))

        move1 = game.get\_player\_move(0)

        move2 = game.get\_player\_move(1)

        if game.bothWent():

            text1 = font.render(move1, 1,  white)

            text2 = font.render(move2, 1,  white)

        else:

            if game.p1Went and p == 0:

                text1 = font.render(move1, 1,  white)

            elif game.p1Went:

                text1 = font.render("Locked in", 1,  Mint\_Green)

            else:

                text1 = font.render("Waiting...", 1,  Mint\_Green)

            if game.p2Went and p == 1:

                text2 = font.render(move2, 1,  white)

            elif game.p2Went:

                text2 = font.render("Locked in", 1,  Mint\_Green)

            else:

                text2 = font.render("Waiting...", 1,  Mint\_Green)

        if p == 1:

            win.blit(text2, (width/4 - text.get\_width()/2, 520))

            win.blit(text1, (width\*0.75 - text.get\_width()/2, 520))

        else:

            win.blit(text1, (width/4 - text.get\_width()/2, 520))

            win.blit(text2, (width\*0.75 - text.get\_width()/2, 520))

        for btn in btns:

            btn.draw(win)

    pygame.display.update()

def draw\_stats\_win(win, stats):

    win.fill(BGCOLOUR)

    header\_font = pygame.font.SysFont("Consolas", 60)

    stats\_font = pygame.font.SysFont("Consolas", 40)

    text = header\_font.render("RPS", 1,  white)

    win.blit(text, (width/4 - text.get\_width()/2, 100))

    text = header\_font.render("Shooter", 1,  white)

    win.blit(text, (width\*0.75 - text.get\_width()/2, 100))

    rps\_total\_games = stats\_font.render(f"Total Games: {stats['rps\_total\_games']}", 1,  white)

    win.blit(rps\_total\_games, (width/4 - rps\_total\_games.get\_width()/2, 200))

    rps\_wins = stats\_font.render(f"Wins: {stats['rps\_wins']}", 1,  white)

    win.blit(rps\_wins, (width/4 - rps\_wins.get\_width()/2, 300))

    rps\_losses = stats\_font.render(f"Losses: {stats['rps\_losses']}", 1,  white)

    win.blit(rps\_losses, (width/4 - rps\_losses.get\_width()/2, 400))

    rps\_win\_rate = stats\_font.render(f"Win Rate: {stats['rps\_win\_rate']}", 1,  white)

    win.blit(rps\_win\_rate, (width/4 - rps\_win\_rate.get\_width()/2, 500))

    shooter\_total\_games = stats\_font.render(f"Total Games: {stats['shooter\_total\_games']}", 1,  white)

    win.blit(shooter\_total\_games, (width\*0.75 - shooter\_total\_games.get\_width()/2, 300))

    shooter\_red\_wins = stats\_font.render(f"Red Wins: {stats['shooter\_red\_wins']}", 1,  white)

    win.blit(shooter\_red\_wins, (width\*0.75 - shooter\_red\_wins.get\_width()/2, 400))

    shooter\_yellow\_wins = stats\_font.render(f"Yellow Wins: {stats['shooter\_yellow\_wins']}", 1,  white)

    win.blit(shooter\_yellow\_wins, (width\*0.75 - shooter\_yellow\_wins.get\_width()/2, 500))

    wordler\_guessed\_words = stats\_font.render(f"Total Guessed Words: {stats['wordler\_guessed\_words']}", 1,  white)

    win.blit(wordler\_guessed\_words, (width / 2 - wordler\_guessed\_words.get\_width() / 2, 600))

    pygame.display.update()

def draw\_shooter\_win(red, yellow, red\_bullets, yellow\_bullets, red\_health, yellow\_health):

    win.blit( SPACE, (0, 0))

    pygame.draw.rect(win,  black,  BORDER)

    win.blit( YELLOW\_SPACESHIP, (yellow.x, yellow.y))

    win.blit( RED\_SPACESHIP, (red.x, red.y))

    # Display health

    red\_health\_text =  HEALTH\_FONT.render(f"Health: {red\_health}", 1,  white)

    yellow\_health\_text =  HEALTH\_FONT.render(f"Health: {yellow\_health}", 1,  white)

    win.blit(red\_health\_text, (width - red\_health\_text.get\_width() - 10, 10))

    win.blit(yellow\_health\_text, (10, 10))

    # Draw bullets

    for bullet in red\_bullets:

        pygame.draw.rect(win,  RED, bullet)

    for bullet in yellow\_bullets:

        pygame.draw.rect(win,  YELLOW, bullet)

    pygame.display.update()

def draw\_winner(text):

    draw\_text =  WINNER\_FONT.render(text, 1,  white)

    win.blit(draw\_text, (width / 2 - draw\_text.get\_width() // 2, height / 2 - draw\_text.get\_height() // 2))

    pygame.display.update()

    pygame.time.delay(5000)

**button.py**

import pygame

from constants import \*

#a special button which changes color on hovering (made for the main menu)

class mainButton():

    def \_\_init\_\_(self, image, pos, text\_input, font, base\_color, hovering\_color):

        self.image = image

        self.x\_pos = pos[0]

        self.y\_pos = pos[1]

        self.font = font

        self.base\_color, self.hovering\_color = base\_color, hovering\_color

        self.text\_input = text\_input

        self.text = self.font.render(self.text\_input, True, self.base\_color)

        if self.image is None:

            self.image = self.text

        self.rect = self.image.get\_rect(center=(self.x\_pos, self.y\_pos))

        self.text\_rect = self.text.get\_rect(center=(self.x\_pos, self.y\_pos))

    def update(self, screen):

        if self.image is not None:

            screen.blit(self.image, self.rect)

        screen.blit(self.text, self.text\_rect)

    def checkForInput(self, position):

        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):

            return True

        return False

    def changeColor(self, position):

        if position[0] in range(self.rect.left, self.rect.right) and position[1] in range(self.rect.top, self.rect.bottom):

            self.text = self.font.render(self.text\_input, True, self.hovering\_color)

        else:

            self.text = self.font.render(self.text\_input, True, self.base\_color)

#a normal button

class Button:

    def \_\_init\_\_(self, text, x, y, color):

        self.text = text

        self.x = x

        self.y = y

        self.color = color

        self.width = 100

        self.height = 50

    def draw(self, win):

        pygame.draw.rect(win, self.color, (self.x, self.y, self.width, self.height))

        font = pygame.font.SysFont("comicsans", 25)

        text = font.render(self.text, 1,  black)

        win.blit(text, (self.x + round(self.width/2) - round(text.get\_width()/2), self.y + round(self.height/2) - round(text.get\_height()/2)))

    def click(self, pos):

        x1 = pos[0]

        y1 = pos[1]

        if self.x <= x1 <= self.x + self.width and self.y <= y1 <= self.y + self.height:

            return True

        else:

            return False

**constants.py**

import pygame

import os

pygame.font.init()

pygame.mixer.init()

pygame.mixer.set\_num\_channels(16)

width = 1280

height = 720

win = pygame.display.set\_mode((width, height))

#colors (r, g, b)

white = (255, 255, 255)

black = (0, 0, 0)

Mint\_Green = (172, 255, 172)

Pale\_Lavender = (199, 184, 234)

Electric\_Blue = (3, 169, 244)

RED = (255, 0, 0)

GREEN = (0, 200, 0)

BLUE = (0, 0, 255)

THE\_OTHER\_BLUE = (70, 196, 207)

YELLOW = (255, 200, 0)

Electric\_Green = (57, 255, 0)

Dark\_Red = (139, 0, 0)

DARKGREY = (40, 40, 40)

LIGHTGREY = (100, 100, 100)

BGCOLOUR = DARKGREY

TILESIZE = 80

GAPSIZE = 10

MARGIN\_X = int((width - (5 \* (TILESIZE + GAPSIZE))) / 2)

MARGIN\_Y = int((height - (6 \* (TILESIZE + GAPSIZE))) / 2)

MARGIN\_X1 = int((width - (5 \* (TILESIZE + GAPSIZE))) \* .15)

MARGIN\_Y1 = int((height - (6 \* (TILESIZE + GAPSIZE))) \* .5)

MARGIN\_X2 = int((width - (5 \* (TILESIZE + GAPSIZE))) \* .85)

MARGIN\_Y2 = int((height - (6 \* (TILESIZE + GAPSIZE))) \* .5)

ROCK\_IMAGE = pygame.image.load(os.path.join('Assets', 'Rock.png'))

PLAYER\_ROCK\_IMAGE = pygame.transform.scale(ROCK\_IMAGE, (width\*0.4, height\*0.4))

OPPONENT\_ROCK\_IMAGE = pygame.transform.flip(pygame.transform.scale(ROCK\_IMAGE, (width\*0.4, height\*0.4)), True, False)

PAPER\_IMAGE = pygame.image.load(os.path.join('Assets', 'Paper.png'))

PLAYER\_PAPER\_IMAGE = pygame.transform.scale(PAPER\_IMAGE, (width\*0.4, height\*0.4))

OPPONENT\_PAPER\_IMAGE = pygame.transform.flip(pygame.transform.scale(PAPER\_IMAGE, (width\*0.4, height\*0.4)), True, False)

SCISSOR\_IMAGE = pygame.image.load(os.path.join('Assets', 'Scissor.png'))

OPPONENT\_SCISSORS\_IMAGE = pygame.transform.scale(SCISSOR\_IMAGE, (width\*0.4, height\*0.4))

PLAYER\_SCISSORS\_IMAGE = pygame.transform.flip(pygame.transform.scale(SCISSOR\_IMAGE, (width\*0.4, height\*0.4)), True, False)

BACKGROUND\_TITLE = pygame.transform.scale(pygame.image.load(os.path.join('Assets', 'TitleBG.png')), (width/6, height\*0.37))

RECTANGLE = pygame.transform.scale(pygame.image.load(os.path.join('Assets', 'Win.png')), (width, height))

BLUFF = ["ROCK", "PAPER", "SCISSORS"]

# Create a partition in the middle

BORDER = pygame.Rect(width // 2 - 5, 0, 10, height)

# Load sounds

BULLET\_HIT\_SOUND = pygame.mixer.Sound(os.path.join('Assets', 'collision.mp3'))

BULLET\_FIRE\_SOUND = pygame.mixer.Sound(os.path.join('Assets', 'smoke\_bomb.mp3'))

WIN\_SOUND = pygame.mixer.Sound(os.path.join('Assets', 'win!.mp3'))

SPACE\_BGM = pygame.mixer.Sound(os.path.join('Assets', 'space\_bgm.mp3'))

# Fonts

HEALTH\_FONT = pygame.font.SysFont('Consolas', 40)

WINNER\_FONT = pygame.font.SysFont('Consolas', 100)

# Game settings

FPS = 60

VEL = 5

BULLET\_VEL = 7

MAX\_BULLETS = 3

SPACESHIP\_width, SPACESHIP\_height = 120, 80

BOX\_width = BOX\_height = 80

# Custom user events

YELLOW\_HIT = pygame.USEREVENT + 1

RED\_HIT = pygame.USEREVENT + 2

# Load and scale spaceship images

YELLOW\_SPACESHIP\_IMAGE = pygame.image.load(os.path.join('Assets', 'spaceship\_yellow.png'))

YELLOW\_SPACESHIP = pygame.transform.rotate(pygame.transform.scale(YELLOW\_SPACESHIP\_IMAGE, (SPACESHIP\_width, SPACESHIP\_height)), 270)

RED\_SPACESHIP\_IMAGE = pygame.image.load(os.path.join('Assets', 'spaceship\_red.png'))

RED\_SPACESHIP = pygame.transform.rotate(pygame.transform.scale(RED\_SPACESHIP\_IMAGE, (SPACESHIP\_width, SPACESHIP\_height)), 90)

# Load and scale background image

SPACE = pygame.transform.scale(pygame.image.load(os.path.join('Assets', 'space.png')), (width, height))

USER\_DATA\_DIR = "user\_data"

MAX\_USERNAME\_LENGTH = 15

MAX\_PASSWORD\_LENGTH = 20

#font

def get\_font(size): # Returns Press-Start-2P in the desired size

    return pygame.font.Font("Assets/font.ttf", size)

**scan\_server.py**

import socket

import threading

# Shared variable to store the found server IP

found\_server\_ip = None

lock = threading.Lock()  # Ensure thread-safe access to shared variables

def get\_local\_ip():

    """ Get the local IP address of the client machine. """

    s = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

    s.connect(("8.8.8.8", 80))

    ip = s.getsockname()[0]

    s.close()

    return ip

def scan\_ip(ip, port):

    """ Helper function to scan a single IP for an open port. """

    global found\_server\_ip

    with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

        s.settimeout(0.1)

        if s.connect\_ex((ip, port)) == 0:

            with lock:

                if not found\_server\_ip:  # Only set if no server is found yet

                    found\_server\_ip = ip

                    print(f"Server found: {found\_server\_ip}:{port}")

def scan\_network\_and\_check\_port(port):

    """ Scan the local network concurrently for active devices on a specified port. """

    global found\_server\_ip

    local\_ip = get\_local\_ip()

    ip\_parts = local\_ip.split('.')

    base\_ip = f"{ip\_parts[0]}.{ip\_parts[1]}.{ip\_parts[2]}."

    found\_server\_ip = None  # Reset the found server IP

    threads = []

    for i in range(1, 255):

        ip = base\_ip + str(i)

        t = threading.Thread(target=scan\_ip, args=(ip, port))

        t.start()

        threads.append(t)

    # Wait for all threads to complete

    for t in threads:

        t.join()

    if found\_server\_ip:

        print(f"Found server at {found\_server\_ip}:{port}")

    else:

        print("No servers found listening on the specified port.")

    return found\_server\_ip

**sprites.py**

import pygame

from constants import \*

class Tile:

    def \_\_init\_\_(self, x, y, letter="", colour=None):

        self.x, self.y = x, y

        self.letter = letter

        self.colour = colour

        self.width, self.height = TILESIZE, TILESIZE

        self.font\_size = int(60 \* (TILESIZE / 100))

        self.create\_font()

        self.rect = pygame.Rect(self.x, self.y, self.width, self.height)  # Rectangle for collision

    def create\_font(self):

        font = pygame.font.SysFont("Consolas", self.font\_size)

        self.render\_letter = font.render(self.letter, True, white)

        self.font\_width, self.font\_height = font.size(self.letter)

    def draw(self, screen, offset\_x=0, offset\_y=0):

        # Update the rect position based on current position

        self.rect.topleft = (self.x + offset\_x, self.y + offset\_y)

        if self.colour is None:

            pygame.draw.rect(screen, white, self.rect, 2)

        else:

            pygame.draw.rect(screen, self.colour, self.rect)

        if self.letter != "":

            self.font\_x = self.x + (self.width / 2) - (self.font\_width / 2) + offset\_x

            self.font\_y = self.y + (self.height / 2) - (self.font\_height / 2) + offset\_y

            letter = pygame.transform.scale(self.render\_letter, (self.font\_width, self.font\_height))

            screen.blit(letter, (self.font\_x, self.font\_y))

class UIElement:

    def \_\_init\_\_(self, x, y, text, colour, font\_size=40):

        self.x, self.y = x, y

        self.text = text

        self.colour = colour

        self.font\_size = font\_size

        self.alpha = 0

        self.create\_font()

    def create\_font(self):

        font = pygame.font.SysFont("Consolas", self.font\_size)

        self.original\_surface = font.render(self.text, True, self.colour)

        self.text\_surface = self.original\_surface.copy()

        # this surface is used to adjust the alpha of the text\_surface

        self.alpha\_surface = pygame.Surface(self.text\_surface.get\_size(), pygame.SRCALPHA)

    def draw(self, screen):

        self.text\_surface = self.original\_surface.copy()

        self.alpha\_surface.fill((255, 255, 255, self.alpha))

        self.text\_surface.blit(self.alpha\_surface, (0, 0), special\_flags=pygame.BLEND\_RGBA\_MULT)

        screen.blit(self.text\_surface, (self.x, self.y))

    def fade\_out(self):

        self.alpha = max(self.alpha - 10, 0)

        self.text\_surface = self.original\_surface.copy()

        self.alpha\_surface.fill((255, 255, 255, self.alpha))

        self.text\_surface.blit(self.alpha\_surface, (0, 0), special\_flags=pygame.BLEND\_RGBA\_MULT)

    def fade\_in(self):

        self.alpha = min(self.alpha + 10, 255)

        self.text\_surface = self.original\_surface.copy()

        self.alpha\_surface.fill((255, 255, 255, self.alpha))

        self.text\_surface.blit(self.alpha\_surface, (0, 0), special\_flags=pygame.BLEND\_RGBA\_MULT)

# Chapter 7 – Game Design

Game design is the study of how to make the game functional by setting game rules, game mechanics, game play which I already briefly discussed in Chapter 1 but now I will go through them thoroughly. I will also discuss more about what I did to make the game more fun rather than visually accepted with the existing resources I have and that is part of game design because even if we make a fancy game with good graphical work if the game design is not well enough it will not attract the user. So I invested a lot of time on doing this section, since without it the game is incomplete.

## Game Rules

**Rock Paper Scissors**

Basic Rules:

* Rock beats Scissors.
* Scissors beat Paper.
* Paper beats Rock.

Health Points:

* Each player starts with a set amount of health points (e.g., 10 HP).
* When you lose a round, your health is deducted based on what move your opponent used:
  + **Rock** deducts **3 HP**.
  + **Scissors** deducts 2 **HP**.
  + **Paper** deduct **2 HP**.

Winning the Game:

* The game continues until one player’s health reaches zero.
* The first player to reduce their opponent’s health to zero wins the game.

**Shooter**Objective:

* Defend your spaceship by shooting energy beams at incoming enemies.
* Survive as long as possible without your health reaching zero.

Health Points:

* Your spaceship starts with **10 health points (HP)**.
* Each time an enemy hits your spaceship, you lose **1 HP**.
* When your health reaches zero, the game is over.

Energy Beams:

* You can shoot a maximum of **3 energy beams at a time**. If all three beams are active, you must wait until one clears (by hitting a target or going off-screen) before firing again.
* Each energy beam deals **1 HP of damage** to enemies.

Winning the Game:

* The game is about achieving the highest score possible by surviving waves of enemies.
* Your final score is recorded when your health reaches zero.

Game Over:

* The game ends when your spaceship’s health reaches zero, or you get overwhelmed by enemies.

**Wordler**

Objective:

* Guess the hidden word within a set number of attempts (e.g., 6 tries per word).
* Each word is a randomly generated 5-letter English word for each round.

Gameplay:

* The player has **unlimited words** to generate, allowing them to start a new round after each game.
* For each round, the player enters a 5-letter word as a guess.

Feedback:

* After each guess, feedback helps the player refine their guesses:
  + **Green**: The letter is in the correct position.
  + **Yellow**: The letter is in the word but in the wrong position.
  + **Gray**: The letter is not in the word at all.
* Use this feedback to make educated guesses and discover the hidden word.

Winning the Game:

* The player wins the round if they guess the word within the allowed attempts.
* A new word can be generated for the player to continue playing as long as they like.

Game Over:

* The game is continuous, with the option to generate a new word at any time.
* If the player fails to guess a word within the allowed attempts, the word is revealed, and the player can start a new round with a different word.

**Game Controls**

**Rock Paper Scissors**

Players can simply click on their choices by double clicking with their mouse. There are no other controls.

**Shooter**

One player can use the **WASD** keys for movement of the spaceship and the other player can use the **Arrow keys.** Respective players can use **Left ctrl and Right ctrl** for shooting energy beams.

**Wordler**

Players can simply type each guesses with their keyboards.

## Assets

The following table shows the list of assets with description I have used to develop this game.

|  |  |  |
| --- | --- | --- |
| Name | Category | Description |
| shoot.mp3 | audio | shooting sound in Shooter |
| collision.mp3 | audio | collision sfx in Shooter |
| win!.mp3 | audio | winning sfx in Shooter |
| space\_bgm.mp3 | audio | background music for Shooter |
| TitleBG.png | image | title in BG in Rock Paper Scissors |
| Win.png | image | serves as a BG layer for win notification |
| spaceship\_yellow.png | image | yellow spaceship |
| spaceship\_red.png | image | red spaceship |
| Rock.png | image | Rock |
| Paper.png | image | Paper |
| Scissor.png | image | Scissors |
| Space.png | image | space background for spaceship |
| font.ttf | font | font for main menu |

## Attempts to make game more fun

Although common mechanics can make the game functional but that’s not enough for entertainment purpose, to entertain the users we need to pay attention to other parts of the game. With keeping that mind set I have included the following features in the game to make the more fun.

* **Letter box Shake:** When player has not entered complete word in wordler, the text box shakes.
* **Letter tiles flip:** When revealing guess output in wordler, the tiles seems to rotate.
* **Background Music:** I have included background music for the game.
* **Sound Effects:** Shooting and collisions sound effects arre added for shooter.
* **Toast Message:** Toast messages notify the player about, connection status, wins and errors. This will keep the user engaged. And informs the player about everything.

# Chapter 8 – Conclusion

## 

## Obstacles

1. Working with a game library completely a new experience for me.

2. It is very sensible work and it demands much time since I a completely new to this. And is the first time I have created a “real” game.

3. I was new to the concept of OOP.

4. It was my first time created a client-server model.

## Achievements

1. Now I know how a game works, each and details that needs to be considered for game development.

2. I got to know more about Object Oriented Programming.

3. Now I have a clear idea on planning a project and executing it.

4. Develop communication skills

5. Growing creative thinking and imagination capability.

## Future Plans

1. To make available a online version where players can from anywhere around the world.

2. To add more sound effects and other rules and features like bluffing in Rock Paper Scissors and many more.

3. To add hint boxes for rules of the game.

4. To link the wordler game to real dictionary and to take real words as input.

5. To add more games meet its name.

# Bibliography

1. Assets : Freepik.
2. SFX : mixit
3. https://docs.python.org/3/tutorial/classes.html
4. <https://www.pygame.org/docs/>
5. <https://wiki.c2.com/?AlanKaysDefinitionOfObjectOriented>
6. <https://asana.com/resources/agile-methodology>
7. <https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/>
8. <https://www.techwithtim.net/>