BSc in Computing in Software Development – Year 4
Gesture-Based User Interface Development

Authors: Krystian Opryszek & Gareth O'Leary

Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

GitHub Repository:

https://github.com/krystianopryszek99/Gesture-Based-UI-Development-Project

May 10, 2022

Contents

1 Purpose of the application

The purpose of this application was to challenge ourselves in developing a gesture-based project using OpenCV and Unity. Python and C# were the programming languages used in the development of this project. During our research we looked at ways we could develop an application with a natural user interface and after discussion, we decided that OpenCV and Unity were the most suitable options. The application is a tennis game, with the player's position-controlled through hand detection and the menus controlled through voice commands.

1.1 User Interface

Menus are a very important aspect of a game. In any game, the first thing a player sees is a menu. Including menus in our game was a key feature that allows the player to freely navigate through the menus. All menus in the game are controlled by voice commands and have the ability to interact as regular buttons using a computer mouse.

The game features of menus:

- Main menu
- Instructions menu
- Pause menu
- Winner menu
- Game over menu

1.1.1 Main Menu

The main menu is very important as it's the first impression of the game. The main menu is designed to be readable and have low visual acuity. The main menu features the following options:

- **Start game** - Starts the game. Brings the player straight into the gameplay.
- **Instructions** - Displays the game instructions. A player can get familiar with the controls of the player and the menus.
- **Quit** - Exits the application.

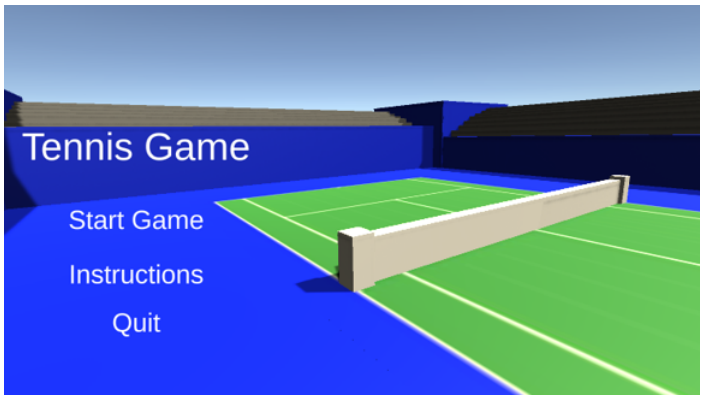


Figure 1: In the main menu of the game – the player can navigate into the gameplay, and game instructions, and exit the game.

1.1.2 Instructions Menu

The game features an instructions menu that steps you through the important controls of the game and menus. The instructions are designed to help the player understand the feeling of the game. The instructions are as follows:

- **Player movement** - Movement of the player by a hand. Required to present 5 fingers when controlling the player.
- **Menus** - Voice commands that allow the player to navigate through the menus.

Figure 2: Game instructions menu – displays the controls of the player and the menus using voice commands.

1.2 In-Game Menus

This game features one in-game menu, which is designed to provide an easy way of navigation through them.

1.2.1 Pause Menu

During the gameplay, the player can pause the game. The pause menu features the following options:

- **Resume** - Returning the player to the gameplay.
- **Main Menu** - This brings the player back to the main menu. This will discard your achievements such as your score.

Figure 3: Pause Menu – allows the player to resume the game and return to the main menu.

1.3 Splash Screens

These are the menus that appear when the player wins or loses. When the player completes a level, he will be taken straight into the next level. A winner menu will appear when the player completes all 3 levels. When a player loses, a game over menu will appear.

1.3.1 Winner Menu

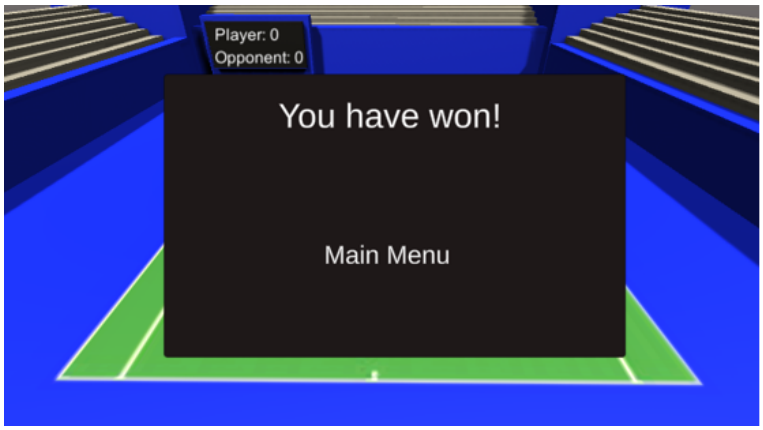


Figure 4: Winner menu – Player can go back to the main menu.

1.3.2 Game over Menu

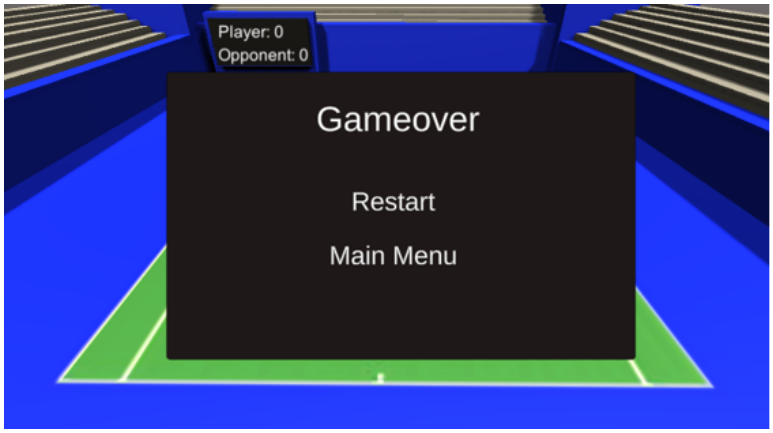


Figure 5: Game over menu – Player can restart the game or go back to the main menu.

1.4 Gameplay

The game starts with relatively easy difficulty. The game is based on a tennis court which was designed to form a “stadium”. The player is challenged to score more than the AI opponent.

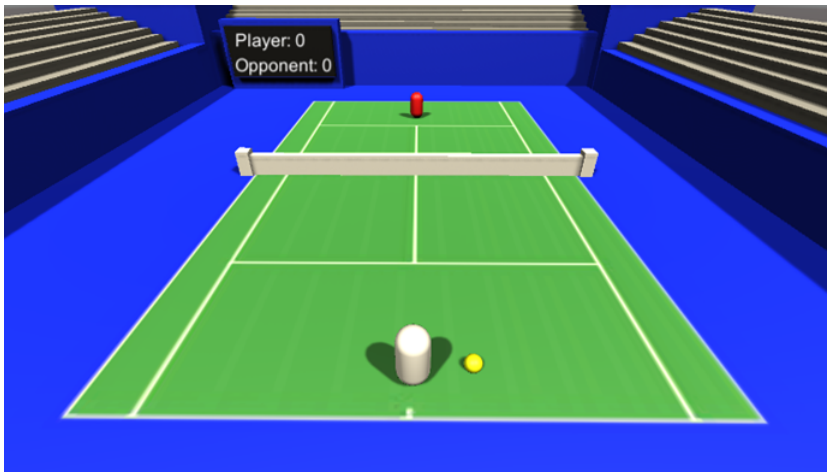


Figure 6: Gameplay – The player plays against an AI opponent.

1.5 Levels

In this game, the levels are designed to start easily, so the player can get used to the controls. The opponent's in-range view and speed are set to low so he can't see the ball from far away. The difficulty of the levels is designed to challenge the player.

All levels are set to 2 for testing so the player can get the feeling of the game and proceed to the next level.

1.5.1 Level 1

- **Score to get** - 5
- **Difficulty** - Easy
- **In-Range** - 6
- **Speed** - 6

1.5.2 Level 2

- **Score to get** - 10
- **Difficulty** - Medium
- **In-Range** - 8
- **Speed** - 7

1.5.3 Level 3

- **Score to get** - 15
- **Difficulty** - Hard
- **In-Range** - 10
- **Speed** - 8

1.6 Testing

Throughout the development of the project, we decided to use testing as it's a process of verifying that an application does what it is supposed to do. In this project, we decided to build a piece of code and test it to see if there were any bugs.

Tennis Game								
ID	**Sprint**	**Test Case Ref**	**Test Case Name**	**Test Case Category**	**Search Parameters / Instructions**	**Expected Result**	**Actual Result**	**Results Pass/Fail**
1.00	1	TC.001	Main Menu	GUI - Voice Command	1. Launch Game 2. Say "Start game"	Loads level 1 and gameplay begins.	Level 1 loads and gameplay begins.	Pass
1.00	1	TC.002	Main Menu	GUI - Voice Command	1. Launch Game 2. Say "Instructions"	Displays instructions menu.	Displays instructions menu.	Pass
1.00	1	TC.003	Main Menu	GUI - Voice Command	1. Launch Game 2. Say "Exit"	Exit the application and print the message in the console.	Exits the game and print the message to console	Pass
2.00	1	TC.004	Instructions Menu	GUI - Voice Command	1. Launch Game 2. Say "Instructions" 3. Say "Back"	Displays the instructions menu and then displays the main menu again.	First displays instructions menu and then the main	Pass
3.00	1	TC.005	Level 1	Player Controls - Hand Gesture	1. Launch Game 2. Say "Start game" 3. Move your hand to the left	Players moves left	Player moves to the left	Pass
3.00	1	TC.006	Level 1	Player Controls - Hand Gesture	1. Launch Game 2. Say "Start game" 3. Move your hand to the right	Players moves right	Player moves to the right	Pass
3.00	1	TC.007	Level 1	Player Controls - Hand Gesture	1. Launch Game 2. Say "Start game" 3. Hit the ball	Hits the ball towards the opponent's side	Ball gets hit to the opponent's side	Pass
3.00	1	TC.008	Level 1	Player Controls - AI	1. Launch Game 2. Say "Start game" 3. Opponent hits the ball back	The opponent hits the ball back to the player	Opponent hits the ball back to the player	Pass
3.00	1	TC.009	Level 1	Scoring	1. Launch Game 2. Say "Start game" 3. Hit the ball and score	Player gets 1 score	Player receives 1 score	Pass

Figure 7: Test Plan.

3.00	1	TC.010	Level 1	Scoring - AI	1. Launch Game 2. Say "Start game" 3. Opponent hits the ball and scores	Opponent gets 1 score	Opponent receives 1 score	Pass
3.00	1	TC.011	Level 1	Scoring	1. Player scores 5 points	Loads level 2	Level 2 loads	Pass
4.00	1	TC.012	Level 2	Scoring / GUI	1. Opponent scores 5 points	Displays game over menu	Game over menu is displayed	Pass
4.00	1	TC.013	Level 2	GUI - Voice Command	1. Repeat step from TC.012 3. Say "Restart"	Restarts the game	Game is restarted	Pass
4.00	1	TC.014	Level 2	GUI - Voice Command	1. Repeat step from TC.012 3. Say "Main Menu"	Displays main menu	Displays the main menu	Pass
4.00	1	TC.015	Level 2	GUI - Voice Command	1. Say "Pause game"	The gameplay gets paused	The game pauses	Pass
4.00	1	TC.016	Level 2	Scoring	1. Player scores 10 points	Loads level 3	Level 3 loads	Pass
5.00	1	TC.017	Level 3	GUI	1. Player scores 15 points	Displays winner menu	Winner menu is displayed	Pass
5.00	1	TC.018	Level 3	GUI - Voice Command	1. Repeat step from TC.017 3. Say "Main Menu"	Displays main menu	Displays the main menu	Pass

Figure 8: Test Plan.

2 Gestures identified as appropriate for this application

Gesture recognition can be seen as a way for computers to begin to understand human body language. Computer vision-based gesture recognition enables people to communicate more naturally with machines. [1] Compared to primitive user interfaces, such as keyboard and mouse, it builds a richer bridge between computers and humans. Gesture control devices recognize and interpret human body movements, allowing the user to interact with a computer system.

Identifying and incorporating gestures that were suited to the style and flow of the game was the first part of the research for this project. The proposed application is a tennis game, looking at ways of how the player could be moved and how the ball could be struck had influencing factors of what gestures could be used. The number one priority when developing this project was to make the interaction comfortable for the player and to make it more realistic by integrating the appropriate gestures. Another question to take into consideration was what gestures could be used effectively with the technologies used in the development of this application. Hand gestures are one of the techniques to communicate with computer vision technologies such as OpenCV and Unity.

Hand gesture recognition can be divided into two categories:

1. Hardware-based.
2. Vision-based.

The game implemented as part of the module is vision-based, meaning image processing techniques and inputs from a computer webcam are used. [2] Hand gesture recognition is one of the most practical and popular solutions for improving human-computer interaction. During the design process of the application, the decision was made that the player would only be moved by hand detection. Without the use of hardware, it would be quite difficult to incorporate hand movements to strike the ball at the opponent. A study by [3] conducted an experiment that counted fingers and recognized gestures using the maximum distance between the centroid of fingers. The following results indicated that finger detection was most effective with either one finger or five fingers(palm) at a 100% success rate. These results were very encouraging and inspired us to continue with the development of the application with OpenCV. During the testing phase, we found our results to be the same as the study by [3], this had a major influence on deciding to use our hand to move the player in the game.

Originally, the plan was to implement a game completely controlled through different types of hand gestures, but after testing the application numerous times, voice recognition was considered a better way to control the menus.

Voice recognition helped increase the performance of the user experience, playing the game requires a lot of concentration, and using multiple hand gestures proved quite difficult. The player can focus more on the hand gestures when using voice commands for the menus. We tested the performance of the game using hand gestures and noticed that the game was much more enjoyable with voice commands implemented.

The different voice commands used in the project:

- **Start** - The start command is executed when the player says phrases such as "Start the game" and "Play the game". This will start the game when initialized.
- **Instructions** - This command is executed when the player says phrases such as "Instructions" and "Show Instructions". This will display a menu on how to operate the game.
- **Exit** - The exit command is executed when the player says phrases such as "Exit now" and "Exit the game".
- **Back** - Phrases such as "Back" and "Go back" when executed will escort the player to the main menu from the instruction menu.
- **Pause** - The pause command will be executed when the player says phrases such as "Pause" and "Pause the game". This action can only be achieved during the gameplay and will pause the game.
- **Resume** - The resume command will be executed when the player says phrases such as "Resume" and "Resume the game". This action will resume the game from the pause menu.
- **Main Menu** - This command will be executed when the player says phrases such as "Main menu" and "Go to the main menu". This will escort the player to the main menu from the pause menu.
- **Restart** - The restart command will be executed when the player says phrases such as "Restart" and "Restart the game". This action will appear on the game over the menu.

3 Hardware used in creating the application

The reasoning behind our choice of hardware was due to us being in different locations. We used two different types of hardware for the development of this project. We used our laptop webcam and microphone.

3.1 Webcam

We used our laptop integrated webcam for OpenCV to detect our hand gestures which allowed accurate hand movements.

Figure 9: Computer integrated webcam.

3.2 Microphone

We also decided to add voice commands in our project for the overall navigation through the menus using our laptop integrated microphone. During the testing phase, we discovered that navigating through the menu while playing the game from a distance was uncomfortable for the player and decided to include voice commands. We decided to add voice commands because they best suited our application in terms of navigating through the menus from a distance.

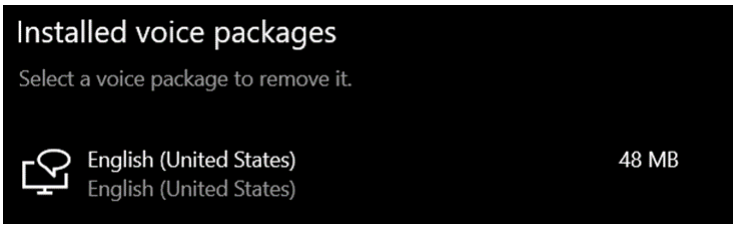


Figure 10: Voice packages

4 Architecture for the solution

This section will give an overview of the libraries used and how the gestures were implemented in the application.

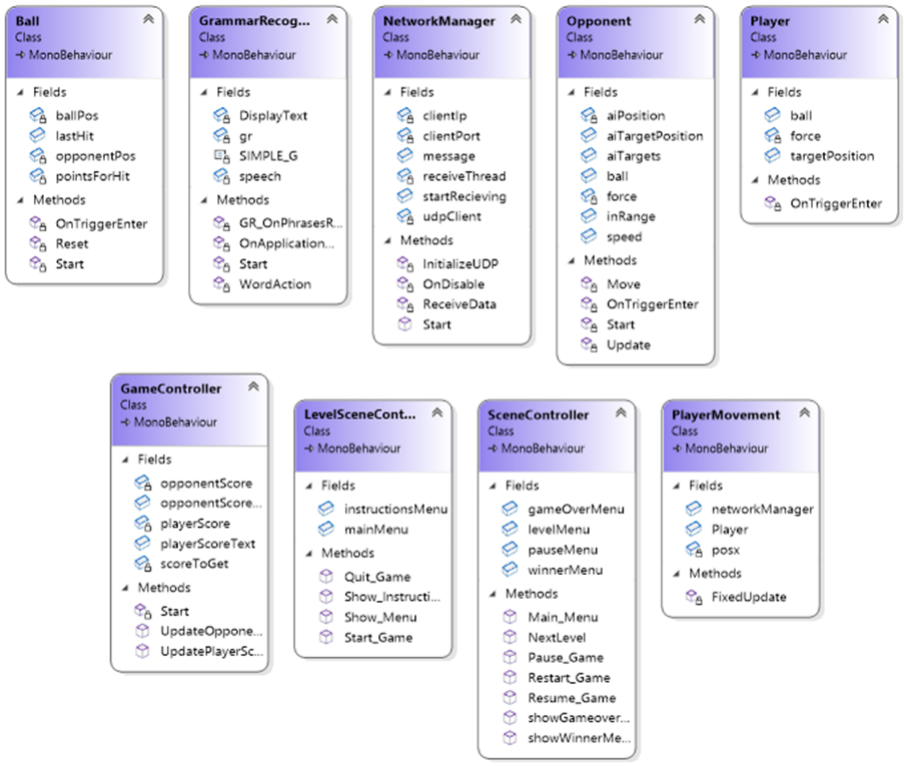


Figure 11: Class Diagram.

4.1 Libraries and Programming languages used

1. **OpenCV** - OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. [4]

2. **MediaPipe** – MediaPipe offers open-source cross-platform, customizable ML solutions for live and streaming media. [5]

3. **C#** - C# is a general-purpose, multi-paradigm programming language.

4. **Python** – Python is a high-level, interpreted, general-purpose programming language.

4.2 How to run the application

To run the application on the server-side, navigate to the python folder and run python Detection.py. This will start the server on the specified port number and IP address. When the server is running, open the unity folder and start the main scene. We used Unity Version 2019.4.18f1 throughout the development of this application.

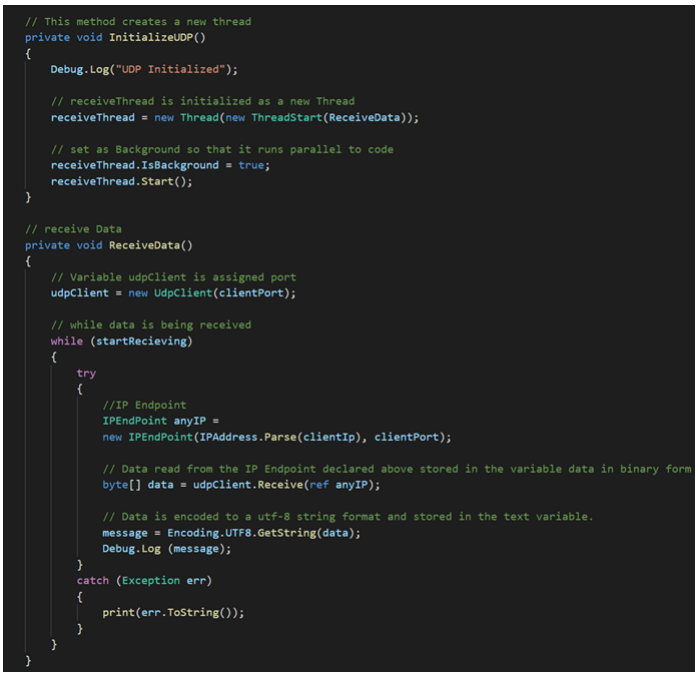
4.3 Communications

Communication between unity and python is possible through sockets. Socket programming is a program that enables two sockets to send and receive data, bi-directionally, at any given moment. [6] The Python script Detection.py was used as a server for sending the data via sockets to the unity client which used User Datagram Protocol (UDP) to receive the data. A UDP socket is a type of computer protocol that is used to transmit and receive information through a network. [7] It was also feasible to use Transmission Control Protocol (TCP) to receive the data in unity, but we decided to use UDP because of its efficiency and how fast it was in comparison to TCP.

```
import socket

# IP
SERVER_IP = "127.0.0.1"
#Port
PORT_NUMBER  = 8080

message = "0"

# Create a client socket
clientSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
clientSock.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
# Connect to the server
clientSock.connect((SERVER_IP,PORT_NUMBER))

print("Server IP:", SERVER_IP)
print("Port Number:", PORT_NUMBER )
```

Figure 12: Socket programming.

Figure 12 displays the setup required for socket programming. The Socket library is required to be imported so that we can use it in our application. We created a socket object that is stored in the "client Sock" variable, this has a constructor that provides a family and type parameter. Once we have an initialized socket object, we can open a connection with the specified port and server IP.



```
// This method creates a new thread
private void InitializeUDP()
{
    Debug.Log("UDP Initialized");

    // receiveThread is initialized as a new Thread
    receiveThread = new Thread(new ThreadStart(ReceiveData));

    // set as Background so that it runs parallel to code
    receiveThread.IsBackground = true;
    receiveThread.Start();
}

// receive Data
private void ReceiveData()
{
    // Variable udpClient is assigned port
    udpClient = new UdpClient(clientPort);

    // while data is being received
    while (startRecieving)
    {
        try
        {
            //IP Endpoint
            IPEndPoint anyIP =
            new IPEndPoint(IPAddress.Parse(clientIp), clientPort);

            // Data read from the IP Endpoint declared above stored in the variable data in binary form
            byte[] data = udpClient.Receive(ref anyIP);

            // Data is encoded to a utf-8 string format and stored in the text variable.
            message = Encoding.UTF8.GetString(data);
            Debug.Log (message);
        }
        catch (Exception err)
        {
            print(err.ToString());
        }
    }
}
```

Figure 13: UDP client.

Figure 13 displays part of the network manager-script used to receive the data sent from the server. In the ReceiveData() method, we created a UDP client and assigned a port to it so we could receive the data. Creating a try-catch block for the UDP connection was done for any exceptions thrown. The data received from the server is in bytes so we encode it to a string and store it in the variable “message” so we can access it.

4.4 Hand Detection

Hand detection is a technique used that allows you to detect different points on your hand. The two Python libraries used in creating a system for hand detection are OpenCV and MediaPipe. OpenCV is used to perform operations associated with computer vision and MediaPipe is used to execute the actual hand detection and tracking of our input image. Figure 14 displays an image of our application running, the points are shown on the image are known as the hand landmarks and are used to identify the hand points.

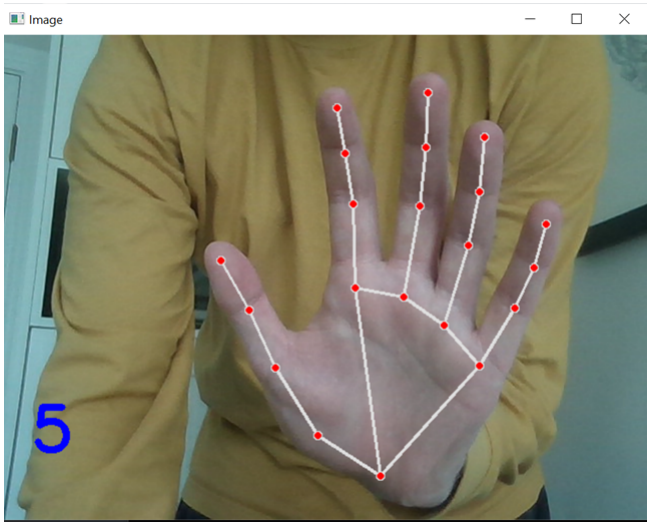


Figure 14: Hand Detection.

The steps are taken to detect the hand landmarks from the image:

1. Initialize `mp.solutions.hands` and `mp.solutions.drawing_utils` - Responsible for detecting and drawing the hand landmark points on the output image.
2. Convert the BGR image to an RBG image.
3. Store the hand landmark detection results in the variable "results" along with the converted image.
4. Check to see if the points are detected or not
5. Create a loop to enable us to work with one hand at a time
6. Loop to get us the hand landmark information which will give us the x and y coordinates of each listed point in Figure 14.
7. Find the height, width, and channel of the image.
8. Find the central positions of the identified hand points.
9. Draw the hand landmarks which can be seen in Figure 14.

The steps labelled above can be seen in Figure 15.

```
imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Process the image and find hands
results = hands.process(img)
# List to store the Landmark's Coordinates
lmList = []
 # checking whether a hand is detected
if results.multi_hand_landmarks:
    hand_lms = results.multi_hand_landmarks[-1]
    for id, lm in enumerate(hand_lms.landmark):
            # find the height, width, and channel of our image
            h, w, c = img.shape
            # get the central positions of the identified hand points.
            centerx, centery = int(lm.x *w), int(lm.y*h)
            message=str((centerx-320)*(5.7/320))
            lmList.append([id, centerx, centery])
        # Drawing the Landmarks for only One Hand
        # Landmarks will be drawn for the Hand which was Detected First
    mpDraw.draw_landmarks(img, hand_lms, mpHands.HAND_CONNECTIONS)
```

Figure 15: Drawing hand landmarks.

4.5 Moving the player with hand detection

1. Socket Connection created.
2. UDP client set up in Unity.
3. Hand detection successful.

Once the three points declared above were complete, the last part to do in relation to hand detection was to be able to move the player. For the game, we only wanted to give the player the ability to move along the x-axis. Previously, we got the central positions of the hand points as seen in Figure 15, we calculated the x points by the distance we wanted to move our hand left and right (length of the court).

```
# get the central positions of the identified hand points.
centerx, centery = int(lm.x *w), int(lm.y*h)
message=str((centerx-320)*(5.7/320))
```

Figure 16: Calculating players position.

The results of the calculation are stored in the variable “message”.

```
# Send message if 5 fingers open
if allFingers == 5:
        clientSock.sendto(bytes(message,'utf-8'), (SERVER_IP, PORT_NUMBER))
```

Figure 17: Sending message.

The message storing the player's position is sent to the UDP client in unity when five fingers are being detected.

```
// get the data from network manager
string data = networkManager.message;

// parse string to float
float.TryParse(data, out posx);

// Player position
Player.transform.position = new Vector3(posx, 1.0f, 10f);
```

Figure 18: Player position.

Finally, in the player movement script, we access the data received from the network manager. The data received from the network manager script is a string, we convert it to a float and set it as the x position of the player. We can now successfully move the player left and right with hand detection.

4.6 Voice Recognition

`GrammarRecogniser.cs` – Listens for words and phrases from XML file.
`Grammar.xml` – stores the grammar words and phrases. The implementation of voice recognition in our project consisted of a script used to recognize grammar incorporated from an XML file. The XML grammar file must be placed in the streaming assets folder of the project.

1. Load XML file.
2. Handle OnPhrasedRecogized event.
3. Start grammar recognizer.

```
private void GR_OnPhrasesRecognised(PhraseRecognizedEventArgs args)
{
    Debug.Log("Recognised Something...");
    // Read the semantic meanings from the args returned
    // Put them in a string to print a message in the console
    StringBuilder message = new StringBuilder();

    SemanticMeaning[] meanings = args.semanticMeanings;

    // Returned a set of name/value pairs - keys/values
    foreach (SemanticMeaning meaning in meanings)
    {
        string keyString = meaning.key.Trim();
        string valueString = meaning.values[0].Trim();

        message.Append("Key: " + keyString + ", Value: " + valueString +
                System.Environment.NewLine);

        speech = valueString;
    }

    DisplayText.text = message.ToString();
    Debug.Log(message);
    WordAction(speech);

}
```

Figure 19: Phrase recognizer.

The `GR_OnPhrasedRecognize()` method is used to recognize the phrases in the XML file. Once the phrase is detected the speech variable is passed into the `wordAction()` method and called. The `wordAction()` method consists of a switch statement to determine what action is performed by the passed in speech variable, this can be seen in Figure 20. Once the phrase is recognized, the specified method is executed and will take the user to the menu stated. There is an if statement in each case so that the phrase called can only be executed from the scene defined.

```
private void WordAction(string speech)
{
    switch(speech)
    {
        case "start":
            if (SceneManager.GetActiveScene () == SceneManager.GetSceneByName ("MainMenu"))
            {
                FindObjectOfType<LevelSceneController>().Start_Game();
            }
            break;
        case "instructions":
            if (SceneManager.GetActiveScene () == SceneManager.GetSceneByName ("MainMenu"))
            {
                FindObjectOfType<LevelSceneController>().Show_Instructions();
            }
            break;
        case "exit":
            if (SceneManager.GetActiveScene () == SceneManager.GetSceneByName ("MainMenu"))
            {
                FindObjectOfType<LevelSceneController>().Quit_Game();
            }
            break;
```

Figure 20: Switch statement to determine which phrases are being used.

5 Conclusions & Recommendations

Evolving gesture user interfaces and new technologies make the future possibilities very exciting, from the original monitors, mouse, and keyboards to voice recognition and hand gesture recognition, which exhibits how much progress has been made. Controlling the player through hand gestures felt more natural, and easier and there was no need to fix problems caused by hardware devices since none is required. Navigating the menus with voice commands made the game more immersive and enhanced the experience for the user. The main aim of the project was to create an application with a natural user interface. The primary goal of the project was to create a system that can identify human-generated gestures and use this information to control the movement of the player and the navigation of the menus. To summarize, the system has accomplished the objectives set out at the beginning of the project and we are both happy with the final product.

5.1 Future Implementations

- **Incorporate more gestures** – After sharing our thoughts on the overall implementation of the game, we felt that more hand gestures could have been incorporated. One idea was to reset the ball by holding up one finger, this would be a nice option for the user.

- **Sound** – Due to time constraints we were unable to implement a sound system in our game. Anyone that has played video games can confirm that sound makes a massive difference to the user experience. Creating

sound for the player striking the ball would have been appropriate, along with the sound of people cheering in the stadium. These enhancements would help make the scenario feel like a real-life game of tennis.

5.2 Limitations

During the implementation of the project, we encountered some issues. Overall the application doesn't contain any bugs which will interrupt the gameplay.

5.2.1 Combining color object detection and hand detection

We found that combining color object detection and hand detection was a major issue. During the testing phase, we discovered that combining two types of messages was a problem. The color object detection was sending the coordinates whereas the hand detection was sending strings in the message. We were successful in combining them together, but we noticed that the feel of the game wasn't as good as it was with just one detection. During the testing phase, we also discovered some issues with the color object detection. The chosen color was sometimes mistaken with colors in the background which caused problems with the player's movement. We decided to pick hand detection as our gesture-based tracking for the player movement as it was more accurate, and the feel of the game was much better compared to color object detection.

5.2.2 Lighting

Although detecting the hand points in the application was very accurate, there were problems when it came to the light environment. If the lighting was poor, detecting the hand proved quite difficult. It is important for the user to play the game in a room that is bright so the webcam can detect the hand.

5.2.3 Restart functionality for levels 2 and 3

There is a small issue with restarting the game in levels 2 and 3. During the testing phase, we discovered that the player could restart the game while being in the gameplay. This was something we didn't want to happen. We decided to write a method that allows the player to restart the game only while being in levels 1,2 and 3. We implemented the following line of code to ensure that.

```
SceneManager.GetActiveScene() == SceneManager.GetSceneByName("Level 1")
```

Figure 21: Code snippets.

Currently, the restart functionality works just for level 1 which is an issue we couldn't fix due to running of the time. We decided to leave the code as shown on figure 21, as we didn't want the player to be affected while playing the game. This issue would be something we would definitely fix first if we were to undertake this project all over again.

5.2.4 Conclusion – Krystian

Throughout the development of the project, I have gained a lot of experience and improved my communication skills while working in a group. From day one, the project has been a challenge and I knew the scope was quite large but I can happily say that we have been very successful in this project. The most valuable thing I have taken from this project was having Python to communicate with Unity.

5.2.5 Conclusion – Gareth

Utilizing OpenCV's library for the first time and learning how we could integrate it with Unity was a challenge but one I greatly enjoyed. It was a great experience to work in a group setting and get familiar with technologies that can adopt different types of gestures. Some of the positives to take away from this experience are working well in a group, knowing how to send and receive data via sockets and have a better understanding of the programming language python.

References

[1] X. Wang, J. Jiang, Y. Wei, L. Kang, and Y. Gao, "Research on gesture recognition method based on computer vision," in *MATEC Web of Conferences*, vol. 232, p. 03042, EDP Sciences, 2018.

[2] T. Thakar, R. Saroj, and V. Bharde, "Hand gesture controlled gaming application," 2021.

[3] M. Perimal, S. Basah, M. Safar, and H. Yazid, "Hand-gesture recognition-algorithm based on finger counting," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 10, no. 1-13, pp. 19–24, 2018.

[4] "Opencv." `https://opencv.org/about/`.

[5] "Mediapipe." `https://mediapipe.dev/`.

[6] "Socket programming in python: Client, server, and peer." `https://www.pubnub.com/blog/socket-programming-in-python-client-server-p2p/`.

[7] "What is a udp socket?." `https://www.easytechjunkie.com/what-is-a-udp-socket.htm`.