2. Implementation

In this chapter we will discuss the implementation process of Playing by Proxy. We will start by giving an overview of how the application functions. Secondly, we will go through all the tools used to make the application possible and how they were used. After this, we will talk about the application's interface on the Client side of the platform. At last, we will delve in more technical terms into each feature of the application.

2.1. Overview

There are two main sides that sustain the operation of Playing by Proxy: the Client side and the Host side. The Blind Player will be situated at the Host side and the Proxy Player at the Client side of the application. When starting the application, both users only have to run the respective bat files, submit the login credentials, and get started. Here we will delve into what happens behind the scenes so the connection between the users is made. Figure 4.1 shows the flow diagram of the relations between both players and the application itself.

To get started in Playing by Proxy, both users, i.e. the Proxy Player and the Blind Player, must acquire a Parsec Gaming account, given that the application is running using Parsec's Software Development Kit (SDK). Both logins are made through a python file that asks for the email and password as inputs and provides a Session ID. The Host login will also provide a Peer ID, this will be the code for the Client to connect to this Host.

The Blind Player will take the role of the Host in the Parsec connection. To get started, they will launch host.exe file with the only argument being their Session ID. After this task is succeeded, the Blind Player will hear a Text-to-Speech voice saying "Welcome to Playing by Proxy". After this, the Host will wait for Client guests. Or, in other words, the Blind Player will wait for the Proxy Player to connect.

The Proxy Player takes the role of the Client in the Parsec connection. After the login, the program will print all the available connections. The available connections refer to the hosting machines of this user's Parsec friends. Here, the Proxy Player chooses the Bind Player's machine to connect, and the connection is successfully done. Now, the Proxy Player will hear the same Text-to-Speech message: "Welcome to Playing by Proxy".

After the connection is made, the Host and the Client will be in constant contact with each other. The first communication they will make is informing each other about their screen's resolutions, so the features that deal with screen coordinates function with precision.

After this, each functionality activated in the Client side will inform the Host side about its activation. The Host side will then do the different operations according to the current functionality being used and the information sent. The technical building of each functionality and how they function can be read at point 4.4 of this chapter, Features.

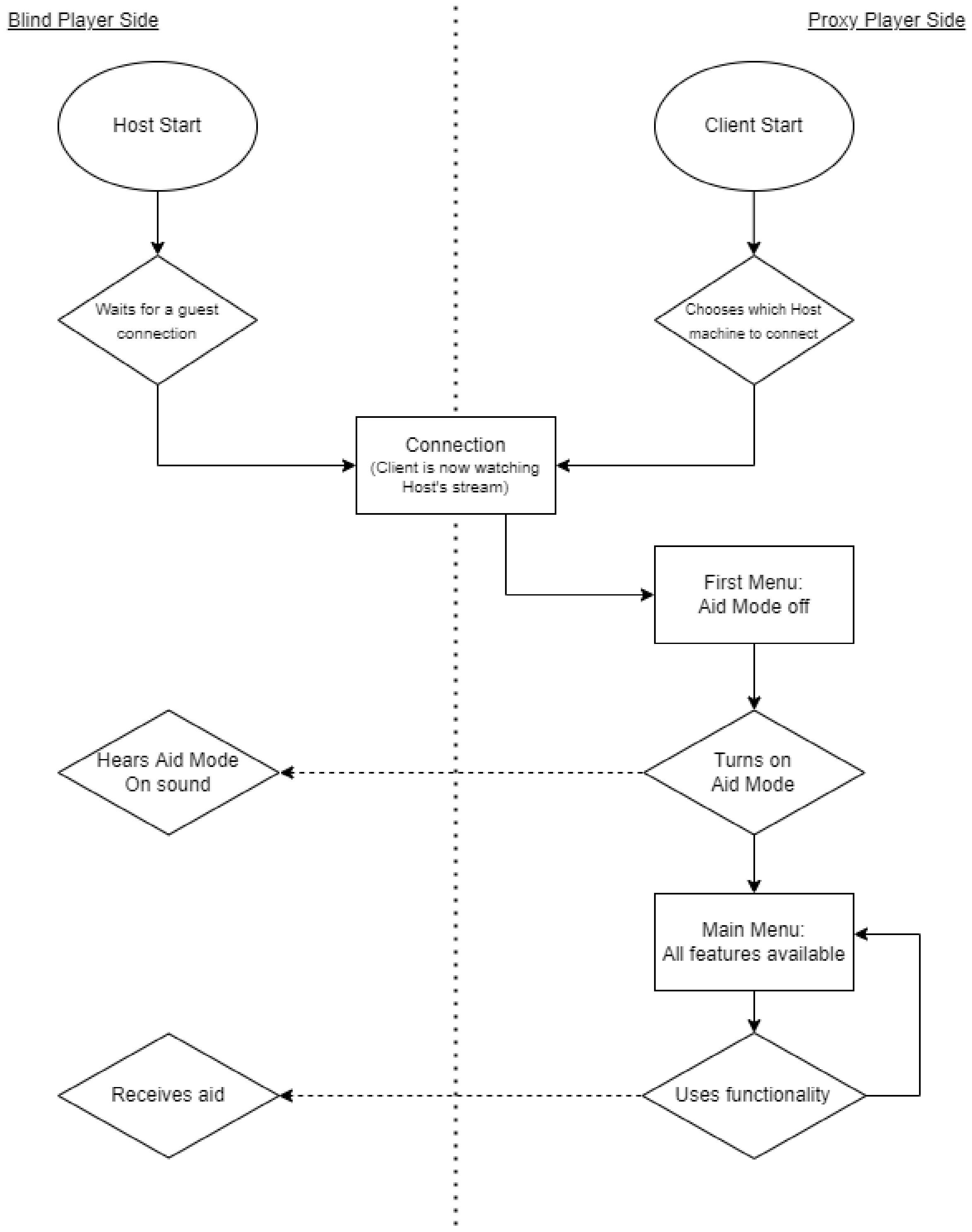

Figure **Error! No text of specified style in document.**.1: Flow Diagram

2.2. Tools

The program of Playing by Proxy is mainly built in C language, using Parsec Gaming's Software Development Kit, but many were the libraries, API's and general tools used to create this desktop streaming application.

The Parsec SDK was the starting point of development of this application. Through compiling and running the Host and Client example files, the first raw connection was reached. The communication between the Host and the Client is made through Parsec Events. Therefore, the Client running cycle will register the Proxy Player's key inputs and send a message to the Host. This message always navigates attached with an ID, so the Host program will recognize its origin and finality and act accordingly.

The Windows API was mainly used when dealing with the screens' resolutions and cursor methods, mainly *GetCursorPos()* and *SetCursorPos()*, to get the current position of the cursor and to set the cursor to a different position, respectively. These methods were used to deal with cursor related features, namely Ping Points, Snap Mouse, and Steer Mouse. Apart from those, the methods *Beep()*, *Sleep()*, and *PlaySound()* were also brought from Windows API. How these last three methods work and how they were used in our context will be explained further on subchapter 4.4 Features.

Simple DirectMedia Layer, or SDL, is the core media library used by Parsec. SDL covers the creation of the application window and deals with the video and audio rendering, as well as the handling of input events.

The graphism built on top of the SDL stream was made with OpenGL. OpenGL is a graphics' library that provides tools and shapes to be drawn on the frames of the computer screen. The menu of the application was entirely drawn using OpenGL. The letters that composed the sentences in the menu were drawn using an OpenGL component called OpenGL Utility (or Glu). Glu was used to write characters from a text bitmap. Methods *writeLetter()* and *writeSentence()* were made to facilitate the writing of the words in the menu.

Two text-to-speech programs were used in the building of this application. All the speech that was written beforehand, such as the Vocal Sounds, was achieved with Amazon Polly, a service that provides clean and lifelike speech. The Press Key mode, Voiced Messages and Labelled Ping Points used eSpeak as the text-to-speech method. eSpeak is a speech synthesiser that functions with console commands, making it an appropriate and easy-to-use choice when dealing with text that will be generated during the use of the application.

2.3. Interface

The application's interface is visible in the Proxy Player's side and only audible in the Blind Player's side. We used OpenGL to draw graphism, on the Proxy Player's side, overlayed on top of the running stream coming from the Blind Player’s computer. This graphism includes the application’s starting menu, the main menu, and all feature menus.

The menus were built with light coloured letters on top of dark rectangle boxes. The rectangles that served as background were achieved using *GL_QUADS*. The colour was set to black and the alpha value to 0.75, representing the opacity of the menu box. The top left vertex of the box was fixed in a point, while the three other vertices depended on the amount of text written in the specific menu. The width of the box was achieved by multiplying the number of letters of the longest sentence by the letter width, plus a short margin. Analogously, the height of the box was achieved by multiplying the number of lines the menu has by the letter height, plus a short margin.

In order to draw text on top of the background boxes, we proceed to the creation of two methods: *writeLetter(letter, x, y)* and *writeSentence(sentence, x, y)*. The *writeLetter(letter, x, y)* method writes a character letter in a position on the screen, using *glutBitmapCharacter()*. The *writeSentence(sentence, x, y)* method proceeds to write a sentence starting on a set position, by using *writeLetter(letter, x, y)* for each letter in the argument sentence and adding the letter width margin space to each letter position.

Navigating through the menu using the F keys will not only activate the features, but also activate the different menus of the application, showing the appropriate background box and text on top.

2.4. Features

All features of this application presented different development challenges, given their varied nature. Ping Points, Snap Mouse, and Steer Mouse dealt, among other issues, with recognizing or resetting Host’s mouse position. Vocal Sounds commanded sound files to be played at a time. Control Sharing involved dealing with Parsecs’ input permissions. Press Key mode made use of eSpeak text-to-speech console program to indicate which keys to be pressed, hold, double clicked, or pressed repeatedly. Voiced Message and Labelled Ping Points required multi-threading programming when asking for user text input, because this had to be synchronous with the main application running cycle.

The features are called and activated in the Client’s side of the program. In order to start using the features, the user, in this case, the Proxy Player, will turn on the Aid Mode. Only then, the application’s features will become available to use. The feature’s modes work as switches, as in when a switch is turned on, all others that would use the same input are turned off. In other words, let’s imagine the Proxy Player is using Snap Mouse to change the Blind Player’s cursor position. Then, they decide to send a Ping Point instead. When they turn on Ping Points mode, preparing to send a Ping Point to the Blind Player, Snap Mouse mode will immediately turn off, so that the program distinguishes clicking on the screen to send a Ping Point to that position or clicking on the screen to snap the Host’s cursor to that same position. Table 4.1 shows the features that require exclusive mouse input or keyboard input and thus cannot be activated at the same time.

Exclusive Mouse Input	Ping Points, Snap Mouse, Control Sharing: Mouse Control, Press Key: Mouse buttons, Labelled Ping Points
Exclusive Keyboard Input	Vocal Sounds, Control Sharing: Keyboard Control, Press Key: Keyboard buttons, Voiced Messages, Labelled Ping Points

Table **Error! No text of specified style in document.**.1: Exclusive Mouse and Keyboard Inputs

On the Host's side, the activation of the features is informed through Parsec Messages that arrive from the Client's side with an ID attached. The ID is asked when the message arrives, and the program will know which feature is being used. Messages may arrive with additional information, such as screen coordinates (x, y) or a text message to be read out loud.

4.4.1 Ping Points

Client Side

The Ping Point mode is activated in the Client Side using F1. While Ping Point is active, clicking anywhere on the screen will register the X and Y values of the position. These values are then sent to the Host side via a Parsec Message. When the connection between the machines of the players is made, the program registers both screen dimensions, so that when coordinates are sent, they are converted to the right resolution. The user may send a new Ping Point while the previous one was still active, replacing the latter.

When a Ping Point is made, there is graphical feedback in the Client's Side. A light blue point is drawn in the position clicked by the Proxy Player, allowing them to visualise the position where the point has been placed. [X AXIS AND Y AXIS]

Pressing *escape* on the Client Side, as well as on the Host Side, will cancel the current Ping Point.

Host Side

In the Host Side, the Ping Point position is registered as soon as the message arrives. While there is an active Ping Point on the screen, the Host's cursor position is being registered, using *GetCursorPos(x, y)*. This method returns a point containing the actual x and y coordinates of the cursor in the Host's screen. The Euclidean distance between the Host's cursor position and the Ping Point position is continuously being calculated.

The two C methods used to create the pings are *Beep(frequency, duration)* and *Sleep(duration)*. *Beep* generates a beep sound with a frequency, measured in hertz, during a duration, measured in milliseconds. *Sleep* makes the program, in this case, the beeps, wait silently during a set duration in milliseconds. This will be the interval of time between each beep sound. In our case, the closer the two points (i.e., the Ping Point position sent by the Proxy Player and the current position of the

Host's cursor) are to each other, the shorter and with higher pitch the beeps will sound, and shorter will be the waiting time between beeps. So, the durations of the beeps and interval times between the beeps are directly proportional to the distance between the two points, while the frequency of the beeps is inversely proportional.

When the Host's cursor gets very close to the Ping Point position, the cursor will snap to that exact position. This happens when the cursor is within a 75px radius from the position in which the Proxy Player placed the Ping Point on. The point is found, a "correct" sound effect is heard, and the Ping Point is done. The Proxy Player may send new points after or during the Host's search. In the latter case, the Ping Point is replaced by the new one.

In the Host side, the program also listens for user input while this mode is activated. If the user presses *escape*, the beeps stop, and the Ping Point is cancelled. If the user presses *enter*, their cursor will go directly to the pretended position, the "correct" sound effect will play, and the Ping Point will be done.

4.4.2 Snap Mouse

Client Side

This mode is turned on by pressing the F2 key. In the Client side, the Snap Mouse mode is akin to Ping Point mode, meaning while this mode is turned on, clicking on the screen with mouse's left click will register the position and send it to the Host via a Parsec Message, with its unique ID. This position is composed of a X value and a Y value, referring to the coordinates on the Client's screen.

Host Side

Because the screen resolution of the Client might not be the same as the Host's, when the Parsec Message arrives, the position is immediately converted to fit Host's screen resolution using the formulas:

X = (Received X * Host Width Resolution) / Client Width Resolution

Y = (Received Y * Host Height Resolution) / Client Height Resolution

The Host's cursor is transposed to the converted X and Y coordinates. The Client will see that, in this moment, the cursors occupy the same position on the screen. The Client may snap the mouse more times while the mode is active. When a snap is made, a beep-like sound is heard on the Host side.

4.4.3 Steer Mouse

Client Side

This mode is turned on by pressing the key F3. When it happens, a message is sent to the Host, informing that this mode has been activated. While Steer Mode is active, pressing down an arrow key on the keyboard will send a message to the Host, containing the according cardinal direction as attachment in the message. Two non-opposite arrow keys may be pressed at the same time, sending both messages to the Host, the intention being to make the Host's cursor move diagonally, according to the two directions sent.

Host Side

In the Host side, a message is received regarding the activation of Steer Mouse mode. While this mode is active, the actual Host cursor position is continuously being registered, using *GetCursorPos(x, y)*. When a message arrives, with an ID referring to a Steer Mouse direction, the Host cursor will move accordingly, using *SetCursorPos(x, y)* method in the following manner:

Direction Received	**Consequent SetCursorPos() arguments**
Right	x + 10, y
Left	x -10, y
Down	x, y + 10
Up	x, y - 10

Table **Error! No text of specified style in document.**.2: How the Host cursor position changes, according to the cardinal direction received.

SetCursorPos(x, y) sets the cursor to a new position. In our case, the cursor will move 10 pixels towards the direction received. In screen coordinates, we have the x horizontal axis, crescent from left to right, and the y vertical axis, crescent from top to bottom. This is why, when the goal is to move the cursor to the right, a value is added to the x, and to move the cursor up, a value is subtracted from the y, and so on.

When two messages regarding a Steer Mouse direction are received in the same cycle iteration, both values are added or subtracted, making the cursor move accordingly. When Steer Mouse mode is deactivated in the Client Side, a message is received, and the program will no longer register Steer Mouse indications.

4.4.4 Vocal Sounds

Client Side

Contrary to the other features, there is no Vocal Sounds mode that needs to be turned on in order to start using the feature. Clicking on F4 will show or hide the complete list of Vocal Sounds available for the Proxy Player to send to the Blind Player. All Vocal Sounds are available to use anytime, including in synchrony with other features that do not use keyboard input (Table 4.1).

The Vocal Sound list also shows which key should be pressed to send a certain sound. For example, the user presses the Y key to send the Vocal Sound “Yes!”, the key G to send “Great!”, or the left arrow key to send “Left!”. The full list of Vocal Sounds can be seen on Figure 3.7. Before sending a Vocal Sound, the program confirms that modes which use keyboard input are turned off, so that when a user is controlling the Blind Player’s keyboard or writing a text message to send a Voiced Message, does not accidentally and without intention send a Vocal Sound.

Host Side

All Vocal Sound audios are present inside a folder amongst the application files. The Host side receives a Parsec Message identified as a Vocal Sound and plays the respective audio file, using the Windows method *PlaySound()*. This method takes as arguments the file name (including its location), a handle to the executable file, set as null, and a flag for playing the sound, in our case set as “file name” because that is how we have identified our sound file.

4.4.5 Control Sharing

Client Side

In the Client Side, the Proxy Player opens their Control Sharing options using the key F5. Doing this, the menu will switch to show the input controlling options and the respective keys to activate them. The user chooses between Mouse Control, Keyboard Control, or Gamepad Control, by pressing the keys M, K, or G, respectively. Various input controls may be activated simultaneously. When turning each control on, a message is sent to the Host side of the application, informing about the input that the Client is requesting to control.

While an input control is being shared, the user may use other features at will (see Table 4.1 to check which features may not be used at the same time as Mouse Control or Keyboard Control). To cancel the control of a before-hand activated input, the user must follow the same steps they did to activate the input. As an example, the user turns Mouse Control on by pressing F5 and then M; the user will turn Mouse Control off by again pressing F5 and then M. This will inform the Host that the input control will no longer be at use by the Client. Alternatively, the Client may receive a Parsec Message by the Host informing that the Blind Player turned off the Control Sharing input themselves.

Host Side

In the Host Side, as soon as an input control initiates the sharing with a Client, an indicative sound is heard, informing the Blind Player about the happening. The Parsec Permission referring to the input in question (i.e., mouse, keyboard, or gamepad) is set to true to the guest who activated it using the Parsec’s SDK ParsecHostSetPermissions() method. This method dictates which Host input controls are permitted for a Client guest to use. The Client can then control the authorised input until that Parsec Permission is set to false again.

If an input control is being shared and the Blind Player presses *escape*, that control stops being shared. A Parsec Message is sent to the Client in case this happens. Alternatively, the Host may receive a message from the Client informing that the sharing of the control has been deactivated. When the input control is no longer being shared, whether by Host or Client deactivation, a closure sound is played to the Host.

4.4.6 Press Key

Client Side

The Proxy Player activates Press Key mode by pressing the key F6. While Press Key mode is activated, pressing a keyboard key or clicking on a mouse button will send a Parsec Message to the Host Side regarding the button that is to be pressed and the manner in which it should be pressed. To prevent sending the same Press Key indication multiple times while a key is being held down, the message is sent only after the key has been released, therefore informing the Host only once.

Press Key mode has four different indications regarding how the Blind Player should press the chosen keyboard key or mouse button: press key, hold key, double press, and press repeatedly. By default, it starts with press key indication. Pressing the key *alt* allows the user to switch between the other Press Key variants, cycling in this order: hold key, double press, press repeatedly, and press key again.

The Parsec Message that will be sent to the Host Side will contain an ID referring to which Press Key variant is currently activated and a String text referring to the button or key that was pressed.

Host Side

In the Host Side of this feature, we make use of eSpeak console commands to vocalise the indication sent by the Client. A command will be sent to the system console containing in its text the Press Key variant that it will use, according to the message received. The message's text, containing the key to be pressed, will be concatenated to the command.

As examples of resulting commands, we can have "espeak -v +f3 Press P", "espeak -v +f3 Hold Space", or "espeak -v +f3 Press Double Press Left Click”. The “-v” component sets the option to select the voice for the synthesised speech. The “f3” component refers to the voice we selected, meaning the third female voice. In this way, all Press Key indications sent by the Proxy Player are vocalised in the Host Side to the Blind Player.

4.4.7 Voiced Message

Client Side

Voiced Message and Labelled Ping Points come together, Client Side, in a single menu called Text Input Message mode. This mode is turned on by pressing F7. While this mode is active, the user can input a text message into the program to later send as a Voiced Message or as a Labelled Ping Point.

Multi-threading was used to capture the user's input synchronously with the program's main running loop. This way, the program can receive and register what the user is typing without interrupting the video and audio threads.

The text input is received using SDL methods and is then registered into a text variable. Upon pressing *enter*, the user sends the text to the Host via a Parsec Message.

Pressing *right* or *left* arrow keys will message the Host informing the Client's intention to change the Voiced Message's language. The two languages available are English and Portuguese.

Host Side

The Host side receives the Parsec Message informing that a Voiced Message has arrived. The attached text is removed from the message, passed as an argument for eSpeak, and is vocalised to the Blind Player.

The default language is set to Portuguese. Receiving a message regarding the change of idiom will change it to English or, if already in English, back again to Portuguese.

4.4.8 Labelled Ping Points

Client Side

The Client Side of Labelled Ping Points is shared with the previous method, Voiced Messages. Being altogether a Text Message Input mode.

The difference being that instead of pressing *enter* to send the written text as a Voiced Message, the user will click anywhere on the screen and send that position as a Labelled Ping Point. The position is registered with a X and a Y value and is sent to the Host via a Parsec Message.

Host Side

In the Host Side, the program recognizes that a Labelled Ping Point has been received and deals with the message almost as if dealing with a regular Ping Point. The Euclidean distance between the received point and the current mouse position is calculated, and the Beeps frequency and duration is calculated all the same (see 4.4.1. Ping Points, Host Side).

With a Labelled Ping Point, the difference lies when the Blind Player reaches the intended point. The program confirms that the Ping Point received was one with a label and, instead of hearing a simple "correct" sound, the Blind Player will hear the text message received by the Proxy Player. As happens with Voiced Messages, the message's text goes through eSpeak synthesiser and is read out loud to the Blind Player. The message's idiom can be Portuguese or English, depending on which the Proxy Player chose it to be.