Jacob Pascoe

17828121

Multiplayer Game Development

Reflective Report

[GitHub Link](https://github.com/Pascoe007/MultiPlayer-GameClient)

Contents

[Introduction 2](#_Toc61341770)

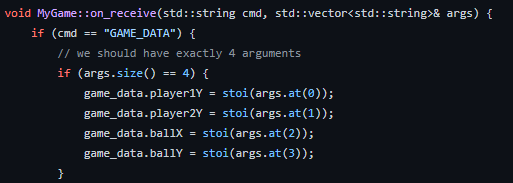
[Implementation 2](#_Toc61341771)

# Introduction

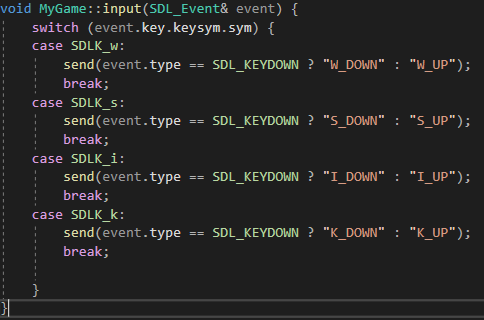
the brief / idea for the client visual design. Include link to YouTube video

# Implementation

the plan for this project was to make a game client for the sever provided for this course. As when the code is given to me was just a single block on the screen, I need to add the player two paddle to the game. This was done by looking at the code that had been given to me. As the ball and player position is being sent from the sever to the client I need to find the code that takes that data and converts it to usable data. As the sever sends the data as a command lines with strings following that. This is handle in the code below. This code has been given to us.



These lines of code take the command line “GAME\_DATA” and breaks it in to smaller checks that can be applied through out the code. In this case after game data there are 4 strings that hold the information for the player one and 2 y position and ball x and y position. Using the “GameData” struct in “MyGame.h” I applied the position for the player two to this Integer in the update function. This could then be used to render the bat on screen by making a cube on screen and setting the position to the player y value. This mean that I could have two paddles on screen that could be moved on the sever. Following this I add the ball using the small code as before. To make the paddles update the paddle on the sever side I had to add commands to send back to the sever. This was done by making a switch statement and calling the send function to send the data from the client to the sever. This was done with the code below. With this code added this means the sever can receive four keyboard inputs done on the client side. The main part to stress about this part is that the client is not handling the movement of the paddles even though the client has keyboard inputs. It sends that data to the sever then is applied to the game and is sent back to the client to update the paddles.



Adding the scores was the next part of this game. This would use the true type font (TTF) add on to SDL this makes it so you can render fonts in to your game easily. with this you have to find a font for your game in this case I used Arial. When getting the font on screen the first thing you have to do is initialise TTF this is done with one line of code which is “TTF\_INIT” then add a error message so you can see if anything goes wrong, all of this should be done before you run the game as you do not want to be loading anything while the game is running. Once the font has been load it need to be passed to the “MyGame.h” script so you can render it. For rendering text you will need a string in this case I converted an integer to a string that had been receive from the sever. You will also need surface this is made by using the font that was loaded earlier and the string also the text colour. This will then be used to make a texture. With this texture you can render it on screen. Below is the code I used for rendering the text on to screen. On line 134 and 135 of this code the surface is being freed, this has to been done other wise the code would fill the users memory up as it will make a new surface every time render is called.

