**Project Development Log – Joseph Henry**

**Prototype 1 Development (The Multiplayer):**

**What this prototype needs to do/have:**

1. Create a client program that will act as the main game loop. This client should define and render a basic display for the game to be displayed within.
2. Create a new file that will run a local server. This file should open a new socket that will listen on a given port for connections. This will then create a connection on a child socket with the client. This connection will be used to send data between the local server and the client.
3. Create a new function that will take in an ip address and port of the server wanting to be joined. It should then take this information and attempt to send data to the requested server address. The game server should show the connection to the host so that they not only know that someone has connected but also who it is (via their ip).
4. Create a parent class of sprite and a child class called player. This will be used to define the character each client will be playing as, initialise them onto the screen and control all the movements. The parent class will also be useful later in the generation of platforms.
5. The local server should be able to hold data from clients in the form of an object.
6. The client should ask the network function to send its player’s object data to the local server. The local server will then send back the other player’s position to the original client. This will allow both clients to have both players positions at all times which means both clients can keep track of the other client’s movements respectively and show them on screen by updating after each communication with the server.

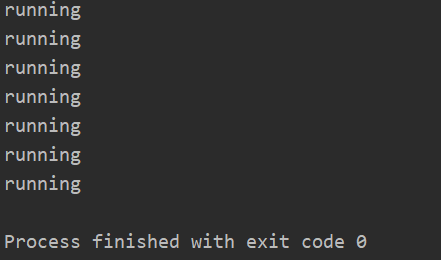
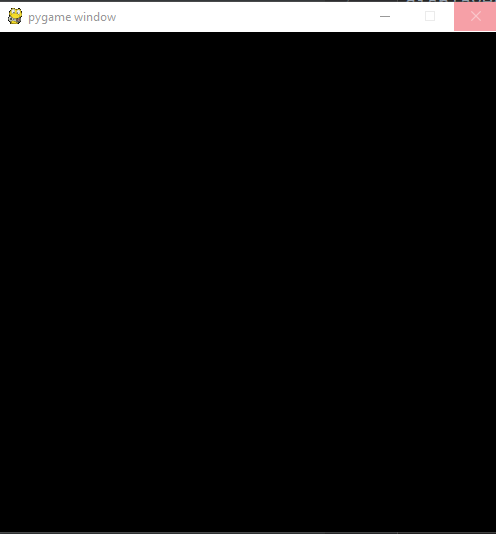
**The Development:**

I first made a basic client program that opened a display window; this will be used to display the game to the players.

import pygame  
displayWidth = 500  
displayHeight = 500

# Creates a display  
gameDisplay = pygame.display.set\_mode((displayWidth, displayHeight))

# This function keeps the game running continuously  
def main():  
 run = True  
 while run:  
 print("running")  
 pygame.quit()  
main()

This worked just as expected by displaying a small box on the screen and printing the word “running” in the console continuously until the code was stopped from running.

I then began work on creating a server for the multiplayer to run off. I did this by opening a socket for the clients to connect to.

import socket

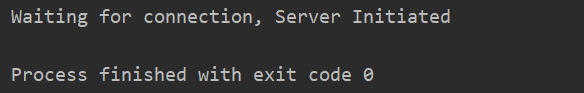
from \_thread import \*

import pickle

# Local IP (IPV4 FROM CMD IPCONFIG, DEVICE SPECIFIC)   
server = socket.gethostbyname(socket.gethostname())  
  
# Server Port  
port = 13010  
  
s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
# The try and except are used to test to see if the port is open. If it is open it will bind the server to the port and if not then the program will print out e to show that the port is in use (e for error).  
try:  
 s.bind((server, port))

except socket.error as e:  
 str(e)

# Puts the socket into listening mode for 2 connections.  
s.listen(2)  
print("Waiting for connection, Server Initiated")

Running this code produced the response “**Waiting for connection, Server Initiated**” in the console and no errors, this means that the port I am using is open and the socket is working fine. It also shows that the port is ready to receive a connection from an external IP.

Then I made a way for the socket to handle incoming connections.

def threaded\_client(client, player):  
 while True:  
 data = pickle.loads(client.recv(2048))  
 # checks to see if any data is being received from the client, if not it assumes that the client is  
 # disconnected and stops running in the background  
 if data:  
 print("connection made")  
  
 else:  
 print("Lost Connection")  
 client.close()  
 break  
  
currentPlayer = 0  
while True:  
 connection, addr = s.accept()  
 print("Connected to:", addr)  
 start\_new\_thread(threaded\_client, (connection, currentPlayer))  
 currentPlayer += 1

This still printed the same text to console (“**Waiting for connection, Server initiated**”) once the file was ran without any errors meaning it was ready to handle any incoming connections.

Next I began work on a method for clients to use to connect to the port.

import socket  
import pickle  
  
  
class clientConnection:  
 def \_\_init\_\_(self):  
 self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.server = "172.16.8.47"  
 self.port = 13010  
 self.addr = (self.server, self.port)  
  
 # connects the players to the server  
 def connect(self):  
 self.client.connect(self.addr)  
 self.client.send(pickle.dumps("getPos"))

This, when called, sends data (“getPos”) to the server which shows the server there is an incoming connection. The server takes the fact that there is data being sent to it to establish the connection.

For this to be used I had to make a small change to the client file so that it would call the clientConnection.

import pygame  
from clientConnection import clientConnection  
  
displayWidth = 500  
displayHeight = 500  
# Creates a display  
gameDisplay = pygame.display.set\_mode((displayWidth, displayHeight))  
# This function keeps the game running continuously  
def main():  
 run = True  
 while run:  
 connection = clientConnection()  
 connection.connect()  
 print("running")  
 pygame.quit()  
main()

Running the server and client at the same time produced the output “connection made” “Connected to: (‘172.16.8.47’, 62478)”. This shows that the server has recognised the incoming connection and established a link to the client.



**Stakeholder Feedback:**

**Prototype 2 Development (Spawn Zone and Physics):**

**What this prototype needs to do/have:**

1. Create better-suited movement controls.
2. I will then need to make a new class called a wall. This will act as the platforms and the floor/roof/walls of the game.
3. Set the walls spawn locations and size and then make the client render the spawn area on the game window by updating the screen.
4. Add all wall sprites to a spritelist; this will make updating or drawing all walls at once easier than by updating the entire sprite list at once rather than each one individually.
5. Use the spritelist to check if any players are colliding with any walls in the spritelist. If yes then stop their movement in that direction. If not then no change.
6. Add gravity to the player so that they do not float but rather fall until they have a platform/floor below them.

**The Development:**

**Stakeholder Feedback:**

**Prototype 3 Development (Map Generation):**

**What this prototype needs to do/have:**

1. Create a title screen so that players are not put straight into playing as soon as the game is launched. This should include a join server button, a start server button and a quit button as well as the title of the game.
2. Make the screen scroll to the right so that more of the map can be shown to the player to navigate through.
3. Generate new platforms procedurally so that there is variety to each play through and so that the map is infinite preventing players from running out of map to run through.
4. Once a platform goes off the viewable area, it should be deleted. This would reduce the chance of the game slowing down due to the large number of entities being stored in local memory.
5. Make holes in the floor at random intervals so that there is some way for the player to actually lose. Also, once the game does end return the player back to the title screen so they can chose whether to start a new game or to quit.

**The Development:**

**Stakeholder Feedback:**