**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. These will be used to define the character each client will be playing as, initialise them onto the screen and create basic movement controls.

1. The local server should be able to hold data from clients in the form of an object.
2. 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.

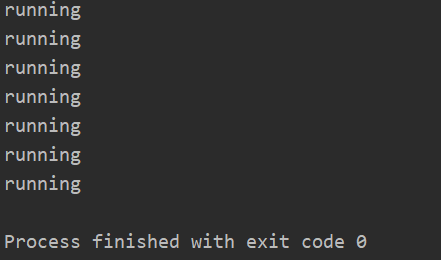
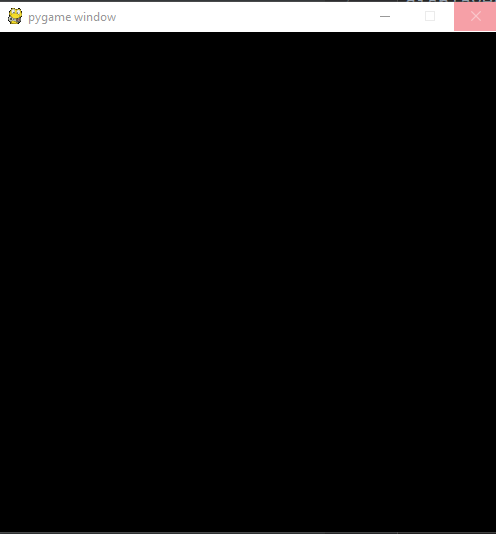
**The Development:**

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

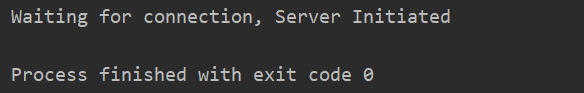
from \_thread import \*

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("Hi"))

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

connection = clientConnection()  
 connection.connect()

while run:  
 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.



Next, I moved onto initialising two player objects and displaying them on the screen. I did this by first creating a class for a general sprite and then a class for the player.

Sprite Class:

import pygame  
  
  
# child class of pygame.sprite.Sprite  
class Sprite(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, x, y, width, height, colour):  
 pygame.sprite.Sprite.\_\_init\_\_(self)  
 self.image = pygame.Surface([width, height])  
 self.image.fill(colour)  
 self.rect = self.image.get\_rect()  
 self.rect.x = x  
 self.rect.y = y

Player Class:

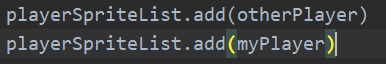
import pygame  
from Sprites import Sprite  
  
  
# Construction of character  
class Player(Sprite):  
 def \_\_init\_\_(self, posargs, width, height, colour):  
 self.x = posargs[0]  
 self.y = posargs[1]  
 Sprite.\_\_init\_\_(self, self.x, self.y, width, height, colour)

Then in the client file, I initialised two separate player sprites.

myPlayer = Player(50, 50, 20, 20, (255, 0, 0))  
otherPlayer = Player(0, 0, 20, 20, (0, 0, 255))

To initialise the player sprites I called the player class, inputting the specific x and y position I want the player to spawn at the width and height of the player and the colour of the player.

To make updating the position of both players easier I added both players’ characters to a sprite list.



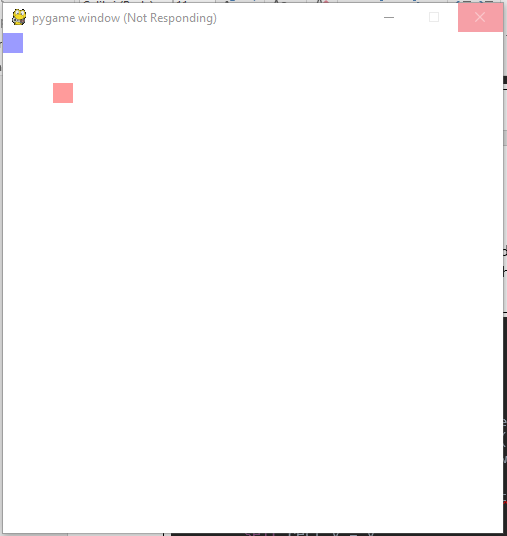
playerSpriteList.draw(gameDisplay)



This allows me to draw all of the player sprites at the same time to the game display.

I also added a clock to set a limit to the number of frames per second that the game can pass.

Running the client now shows the two characters being displayed on a white background, which is just as intended for this stage.



Furthermore, I added a way for the game to handle quitting so that it didn’t crash upon trying to close.

This was a simple addition of just using the inbuilt pygame event checker

for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 run = False

Next, I needed to create player movement controls so that the player can move their character around the screen. For this I first defined a set movement speed for the characters inside of their class as self.vel and set this to have value 3.

This meant that I could now add simple movements to my player objects by changing the x and y positions of the player with simple addition and subtraction of self.vel with movement corresponding to the direction of the keys wasd with s being the centre.

# defines movement controls

def move(self):  
 keys = pygame.key.get\_pressed()  
  
 if keys[pygame.K\_d]:  
 self.rect.x += self.vel  
  
 if keys[pygame.K\_a]:  
 self.rect.x -= self.vel  
  
 if keys[pygame.K\_w]:

self.rect.y -= self.vel

if keys[pygame.K\_s]:

self.rect.y += self.vel

I then also had to add a way that the client would check for these movements being pressed so inside of the main game loop I called the move method for myPlayer.

myPlayer.move()

These movement controls are shown working in this video: [**https://i.gyazo.com/23fccb4f8f13caec2e27fb57ca6c4281.gif**](https://i.gyazo.com/23fccb4f8f13caec2e27fb57ca6c4281.gif)

Now to complete this prototype all I have left to do is to synchronize the two clients together by making them exchange data via the server.

To begin with on the server I made it so the start positions of both players are stored in a list. This would mean that the server has a way to track what the position of each player is by updating the list each time data is received. I then made a way for the client to ask for the original position of its own player by the client sending the message “getPos”. Upon the message “getPos” being found by the server the server would send back the position data of the player whose client is requesting the data.

playerPositions = [[200, 440], [200, 440]]  
  
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:  
 if data == "getPos":  
 reply = (playerPositions[player])

This meant that when the client sent the message “getPos” to the server the server would send back their player’s original position. Then the main information swapping can begin.

I then also changed the clientConnection file so that upon connection the clientConnection sends the message “getPos” so that the client can receive its own starting position. To make the client receive back the information I had to add a new method to the clientConnection class which would return any information received from the server by passing it back to the connect method.

# connects the players to the server  
def connect(self):  
 self.client.connect(self.addr)  
 self.client.send(pickle.dumps("getPos"))  
 return self.getPos()  
  
# returns the position of the object  
def getPos(self):  
 return pickle.loads(self.client.recv(2048))

Since the connect method should originally be called from the client, the return statement returns the data received from self.getPos() back to the client.

In the client file I now had to make the client store what has been returned to it upon connection as its own players position.

I did this by simply making myPlayerPosition = connection.connect and changing how the players are first initialised to use myPlayerPosition and otherPlayerPosition as its x and y coordinates.

def main():  
 # Upon connection receives player position from server so this is just used to initialise the other player  
 otherPlayerPosition = (0, 0)  
  
 pygame.init()  
  
 connection = clientConnection()  
 myPlayerPosition = connection.connect()  
  
 myPlayer = Player(myPlayerPosition, 20, 20, (255, 0, 0))  
 otherPlayer = Player(otherPlayerPosition, 20, 20, (0, 0, 255))

During the main game loop I also had to make the clients swap player position data. So to do this, first I had to create a way for the player to find its own player position data. I did this by first adding a new method to the player class called getPos.

def getPos(self):  
 return [self.rect.x, self.rect.y]

This would simply return the x and y coordinates of the sprite.

Then I had to make a method in the clientConnection for the client to be able to send data to the server in exchange for data back.

# sends data to the server and receives data other client data  
def send(self, data):  
 try:  
 self.client.send(pickle.dumps(data))  
 return pickle.loads(self.client.recv(2048))  
 except socket.error as e:  
 print(e)

This would send data given by the client to the server and return any data given back to the clientConnection from the server back to the client.

Once the otherPlayerPosition is received from the server, I needed a method that I could use in the player class to set the position of the other player’s character on the current clients screen.

To do this I added to the player class the “setPos” method.

def setPos(self, pos):  
 self.rect.x = pos[0]  
 self.rect.y = pos[1]

This will take in the desired positions received back from the server (that is in a list) and split it into x and y, Then set the other players position (x and y) to these.

Finally, on the client I had to get it to run through this cycle every time the game loop restarted and this completed the exchanging of data between clients.

myPlayerPosition = myPlayer.getPos()  
otherPlayerPosition = connection.send(myPlayerPosition)  
otherPlayer.setPos(otherPlayerPosition)

To test this I connected two clients to the server and tried moving on one client to see what it showed on the other, then switched which client I was moving to show both work.

[**https://i.gyazo.com/18482b673707ea7922b379dcbea9092d.gif**](https://i.gyazo.com/18482b673707ea7922b379dcbea9092d.gif)

This video shows that player movements are shown on both screens (red character is that client’s player).

**Final Code:**

**Client File**

import pygame  
from clientConnection import clientConnection  
from Player import Player  
  
displayWidth = 500  
displayHeight = 500  
# Creates a display  
gameDisplay = pygame.display.set\_mode((displayWidth, displayHeight))  
  
playerSpriteList = pygame.sprite.Group()  
  
  
# This function keeps the game running continuously  
def main():  
 # Upon connection receives player position from server so this is just used to initialise the other player  
 otherPlayerPosition = (0, 0)  
  
 pygame.init()  
# begins the connection to the server  
 connection = clientConnection()  
 myPlayerPosition = connection.connect()

# Initialises player sprites  
 myPlayer = Player(myPlayerPosition, 20, 20, (255, 0, 0))  
 otherPlayer = Player(otherPlayerPosition, 20, 20, (0, 0, 255))  
  
 playerSpriteList.add(otherPlayer)  
 playerSpriteList.add(myPlayer)  
  
 clock = pygame.time.Clock()  
  
 run = True  
 while run:

# gets the position data of the other player in exchange for their own position.  
 myPlayerPosition = myPlayer.getPos()  
 otherPlayerPosition = connection.send(myPlayerPosition)  
 otherPlayer.setPos(otherPlayerPosition)  
  
 playerSpriteList.update()  
# Handles exiting game  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 run = False  
# Draws the players to the display  
 gameDisplay.fill((255, 255, 255))  
 playerSpriteList.draw(gameDisplay)  
  
 myPlayer.move()  
 pygame.display.update()  
  
 # sets max fps  
 clock.tick(60)  
  
 pygame.quit()  
 exit()  
  
  
main()

**Stakeholder Feedback:**

**clientConnection File**

import socket  
import pickle  
  
  
class clientConnection:  
 def \_\_init\_\_(self):  
 self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.server = "192.168.0.37"  
 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"))  
 return self.getPos()  
  
 # returns the position of the object  
 def getPos(self):  
 return pickle.loads(self.client.recv(2048))  
  
 # sends data to the server and receives data other client data  
 def send(self, data):  
 try:  
 self.client.send(pickle.dumps(data))  
 return pickle.loads(self.client.recv(2048))  
 except socket.error as e:  
 print(e)

**Sprites file**

import pygame  
  
  
# child class of pygame.sprite.Sprite  
class Sprite(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, x, y, width, height, colour):  
 pygame.sprite.Sprite.\_\_init\_\_(self)  
 self.image = pygame.Surface([width, height])  
 self.image.fill(colour)  
 self.rect = self.image.get\_rect()  
 self.rect.x = x  
 self.rect.y = y

**Server File**

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")  
  
playerPositions = [[200, 440], [200, 440]]  
  
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:  
 if data == "getPos":  
 reply = (playerPositions[player])  
  
 # updates the playerPositions based on which player the data is received from  
 else:  
 if player == 0:  
 playerPositions[0] = data  
 reply = playerPositions[1]  
 else:  
 playerPositions[1] = data  
 reply = playerPositions[0]  
  
 # closes connection if no data is received  
 else:  
 print("Lost Connection")  
 client.close()  
 break  
  
 # sends data back to all open threads  
 client.sendall(pickle.dumps(reply))  
  
  
# keeps track of which player is which connection  
currentPlayer = 0  
while True:  
 connection, addr = s.accept()  
 print("Connected to:", addr)  
 start\_new\_thread(threaded\_client, (connection, currentPlayer))  
 currentPlayer += 1

**Player File**

import pygame  
from Sprites import Sprite  
  
  
# Construction of character  
class Player(Sprite):  
 def \_\_init\_\_(self, posargs, width, height, colour):  
 self.x = posargs[0]  
 self.y = posargs[1]  
 Sprite.\_\_init\_\_(self, self.x, self.y, width, height, colour)  
 self.vel = 3

# Defines movement controls  
 def move(self):  
 keys = pygame.key.get\_pressed()  
  
 if keys[pygame.K\_d]:  
 self.rect.x += self.vel  
  
 if keys[pygame.K\_a]:  
 self.rect.x -= self.vel  
  
 if keys[pygame.K\_w]:  
 self.rect.y -= self.vel  
  
 if keys[pygame.K\_s]:  
 self.rect.y += self.vel  
# sets the position of the player to the given position  
 def setPos(self, pos):  
 self.rect.x = pos[0]  
 self.rect.y = pos[1]  
# returns the position of the player  
 def getPos(self):  
 return [self.rect.x, self.rect.y]

**Stakeholder Feedback:**

All stakeholders tested the prototype and found no bugs so were very pleased with the game in its current state

No additions were required for this prototype.

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

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

1. Create better-suited movement controls. (jumping)
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:**

The first part of this prototype I decided to tackle was the spawn area.

I began this by first creating a new class called wall which would act as the platforms, walls, floors and roofs of the map.

from Sprites import Sprite  
  
# child class of Sprite  
class Wall(Sprite):  
 def \_\_init\_\_(self, x, y, width, height, colour):  
 Sprite.\_\_init\_\_(self, x, y, width, height, colour)

Having it in a separate class mean that I can differentiate it easily from the player class or the parent class, for example to add a new method to the wall that the player or any other classes should not have.

Then in the client I initialised three Wall sprites, a roof, a back wall and the floor.

As the spawn area should be the same on both clients I decided to just generate it purely on the client. This also is more efficient as, for instance, the clients do not have to wait to receive information from the server if it was stored on the server.

# Initialises Spawn Area  
startFloor = Wall(0, 480, 500, 20, (0, 0, 0))  
startWall = Wall(0, 0, 20, 480, (0, 0, 0))  
startRoof = Wall(0, 0, 480, 20, (0, 0, 0))

Initially, it did not recognise Wall as a class as I forgot to import it, but this was a simple fix. By adding “from Wall import Wall” to the top of the client file, it was able to recognise Wall as a class.

Next I decided to add the Walls to a new spritelist, this way I can draw them all at the same time as well as check if the player is colliding with any of the Walls at once.

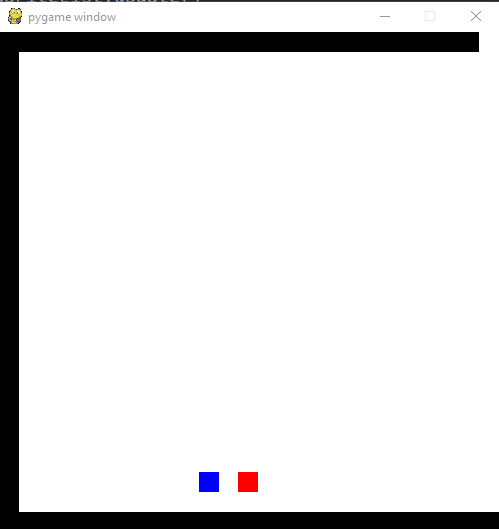
platformSpriteList = pygame.sprite.Group()

# Adds spawn walls to sprite list  
platformSpriteList.add(startFloor)  
platformSpriteList.add(startWall)  
platformSpriteList.add(startRoof)

Finally inside the main game loop, I made the program draw all the sprites in platformSpriteList to the game display.

platformSpriteList.draw(gameDisplay)

Running the game now shows:



The two player sprites are drawn to the game display.

The spawn area is drawn to the game display.

Then, a basic gravity effect was applied to the player.

Inside the player class I added a new attribute called self.gravity which I set to the constant of 5.

However, I only wanted gravity to be applied if the player wasn’t colliding with any platforms below it, so I added a new method to the player class to be called by the game later on.

def notCollide(self):  
 self.rect.y += self.gravity

This function would be called constantly as long as the player is not colliding with a platform.

Next, I decided to implement a collision detection system.

This was simpler than I first anticipated, first in the client I asked whether the current clients player was colliding with any of the sprites inside the platformSpriteList.

if pygame.sprite.spritecollideany(myPlayer, platformSpriteList) is None:  
 myPlayer.notCollide()

This script tests whether one sprite is colliding with any sprite in a spritelist (in this case, whether myPlayer is colliding with any sprite in platformSpriteList) and then returns either “None” if there is no collisions or returns the sprite which is being collided with in the case that there is a collision.

Since the notCollide method only needs to be called when there is not a collision, the if statement on runs if the returned value is None.

This made the gameplay change to this effect.

<https://i.gyazo.com/7a7d1f39f6be2d7f287c379feba9b8b3.gif>

This shows that the player object falls until it is colliding with a platform below it.

Since the player now has gravity I got rid of the s to move down in player.move() as it shouldn’t be needed.

Next, I changed player movements to include jumping rather than to generally move up.

First, I had to add a new attribute to the player class called upVel, which will act as a separate velocity to the self.vel.

self.upVel = 10

I also had to check whether the player was already falling or not so I added a new if statement to the client file that does this

elif pygame.sprite.spritecollideany(myPlayer, platformSpriteList) is not None:  
 myPlayer.falling = False

As well as a new attribute to Player that keeps track of if the player is falling or not

self.falling = False

if not self.jumping and not self.falling:  
 if keys[pygame.K\_w]:  
 if self.upVel == 10:  
 for i in range(12):  
 self.rect.y -= self.upVel  
 self.upVel \* 0.8  
 self.update()  
  
 else: # This will execute if our jump is finished  
 # Resetting our Variables  
 self.upVel = 10  
 self.falling = False  
 self.jumping = False  
 if self.upVel == 10:  
 for i in range(12):  
 self.rect.y -= self.upVel  
 self.upVel \* 0.8  
 self.update()

Now when the player presses the w key, there character is moved up by a value and then gravity brings it back down to the nearest platform.

For example in this clip:

<https://i.gyazo.com/25325bea0fb2b90638c472494ea98b80.gif>

**Final Code:**

**Client File**

import pygame  
from clientConnection import clientConnection  
from Player import Player  
from Wall import Wall  
  
displayWidth = 500  
displayHeight = 500  
  
# Creates a display  
gameDisplay = pygame.display.set\_mode((displayWidth, displayHeight))  
  
platformSpriteList = pygame.sprite.Group()  
playerSpriteList = pygame.sprite.Group()  
  
  
def reDraw(platformSpriteList, playerSpriteList):  
 gameDisplay.fill((255, 255, 255))  
 platformSpriteList.draw(gameDisplay)  
 playerSpriteList.draw(gameDisplay)  
 pygame.display.update()  
  
  
# This function keeps the game running continuously  
def main():  
 # Upon connection receives player position from server so this is just used to initialise the other player  
 otherPlayerPosition = (0, 0)  
  
 pygame.init()  
  
 connection = clientConnection()  
 myPlayerPosition = connection.connect()  
  
 # Initialises player sprites  
 myPlayer = Player(myPlayerPosition, 20, 20, (255, 0, 0))  
 otherPlayer = Player(otherPlayerPosition, 20, 20, (0, 0, 255))  
  
 # Initialises Spawn Area  
 startFloor = Wall(0, 480, 500, 20, (0, 0, 0))  
 startWall = Wall(0, 0, 20, 480, (0, 0, 0))  
 startRoof = Wall(0, 0, 480, 20, (0, 0, 0))  
  
 # Adds spawn walls to sprite list  
 platformSpriteList.add(startFloor)  
 platformSpriteList.add(startWall)  
 platformSpriteList.add(startRoof)  
  
 playerSpriteList.add(otherPlayer)  
 playerSpriteList.add(myPlayer)  
  
 clock = pygame.time.Clock()  
  
 run = True

while run:  
 myPlayerPosition = myPlayer.getPos()  
 otherPlayerPosition = connection.send(myPlayerPosition)  
 otherPlayer.setPos(otherPlayerPosition)  
  
 # updates the players and platforms  
 platformSpriteList.update()  
 playerSpriteList.update()  
  
 if pygame.sprite.spritecollideany(myPlayer, platformSpriteList) is None:  
 myPlayer.notCollide()  
  
 elif pygame.sprite.spritecollideany(myPlayer, platformSpriteList) is not None:  
 myPlayer.falling = False  
  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 run = False  
  
 reDraw(platformSpriteList, playerSpriteList)  
  
 myPlayer.move()  
  
 # sets max fps  
 clock.tick(60)  
  
 pygame.quit()  
 exit()  
  
  
main()

**Wall File**

from Sprites import Sprite  
  
  
class Wall(Sprite):  
 def \_\_init\_\_(self, x, y, width, height, colour):  
 Sprite.\_\_init\_\_(self, x, y, width, height, colour)

**Server File**

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")  
  
playerPositions = [[200, 440], [200, 440]]  
  
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:  
 if data == "getPos":  
 reply = (playerPositions[player])  
  
 # updates the playerPositions based on which player the data is received from  
 else:  
 if player == 0:  
 playerPositions[0] = data  
 reply = playerPositions[1]  
 else:  
 playerPositions[1] = data  
 reply = playerPositions[0]  
  
 # closes connection if no data is received  
 else:  
 print("Lost Connection")  
 client.close()  
 break  
  
 # sends data back to all open threads  
 client.sendall(pickle.dumps(reply))  
  
  
# keeps track of which player is which connection  
currentPlayer = 0  
while True:  
 connection, addr = s.accept()  
 print("Connected to:", addr)  
 start\_new\_thread(threaded\_client, (connection, currentPlayer))  
 currentPlayer += 1

**clientConnection File**

import socket  
import pickle  
  
  
class clientConnection:  
 def \_\_init\_\_(self):  
 self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.server = "192.168.0.37"  
 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"))  
 return self.getPos()  
  
 # returns the position of the object  
 def getPos(self):  
 return pickle.loads(self.client.recv(2048))  
  
 # sends data to the server and receives data other client data  
 def send(self, data):  
 try:  
 self.client.send(pickle.dumps(data))  
 return pickle.loads(self.client.recv(2048))  
 except socket.error as e:  
 print(e)

**Sprites File**

import pygame  
  
  
# child class of pygame.sprite.Sprite  
class Sprite(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, x, y, width, height, colour):  
 pygame.sprite.Sprite.\_\_init\_\_(self)  
 self.image = pygame.Surface([width, height])  
 self.image.fill(colour)  
 self.rect = self.image.get\_rect()  
 self.rect.x = x  
 self.rect.y = y

**Player File**

import pygame  
from Sprites import Sprite  
  
  
# Construction of character  
class Player(Sprite):  
 def \_\_init\_\_(self, posargs, width, height, colour):  
 self.x = posargs[0]  
 self.y = posargs[1]  
 Sprite.\_\_init\_\_(self, self.x, self.y, width, height, colour)  
 self.vel = 3  
 self.upVel = 10  
 self.jumping = False  
 self.gravity = 5  
 self.falling = False  
  
 def move(self):  
 keys = pygame.key.get\_pressed()  
  
 if keys[pygame.K\_d]:  
 self.rect.x += self.vel  
  
 if keys[pygame.K\_a]:  
 self.rect.x -= self.vel  
  
 if not self.jumping and not self.falling:  
 if keys[pygame.K\_w]:  
 if self.upVel == 10:  
 for i in range(12):  
 self.rect.y -= self.upVel  
 self.upVel \* 0.8  
 self.update()  
  
 else: # This will execute if our jump is finished  
 # Resetting our Variables  
 self.upVel = 10  
 self.falling = False  
 self.jumping = False  
 if self.upVel == 10:  
 for i in range(12):  
 self.rect.y -= self.upVel  
 self.upVel \* 0.8  
 self.update()  
  
 def notCollide(self):  
 self.rect.y += self.gravity  
 self.falling = True  
  
 def setPos(self, pos):  
 self.rect.x = pos[0]  
 self.rect.y = pos[1]  
  
 def getPos(self):  
 return [self.rect.x, self.rect.y]

**Stakeholder Feedback:**

After testing this prototype, stakeholders found no bugs again so were happy with the program. However, one piece of feedback was that the jumping animation could be better as it did not show the movement up.

**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:**

To be able to implement a title screen I would need buttons so I first made a new class called Button.

import pygame  
  
red = (200, 0, 0)  
green = (0, 200, 0)  
  
bright\_red = (255, 0, 0)  
bright\_green = (0, 255, 0)

# defines the class button  
class Button:  
 def \_\_init\_\_(self, function, surface, posx, posy, width, height, colour):  
 self.rect = pygame.Rect(posx, posy, width, height)  
 self.colour = colour  
 self.surface = surface  
 self.function = function  
 pygame.draw.rect(self.surface, self.colour, self.rect)

# checks if the mouse is over the button, if the mouse is clicked while hovering over the button, the button completes a task set upon the instanciation of the button.  
 def mouseCheck(self):  
 mouse = pygame.mouse.get\_pos()  
  
 if self.rect.x + self.rect.width > mouse[0] > self.rect.x and self.rect.y + self.rect.height > mouse[1] > self.rect.y:  
 pygame.draw.rect(self.surface, bright\_green, self.rect)  
  
 if pygame.mouse.get\_pressed()[0]:  
 self.function()  
  
 else:  
 pygame.draw.rect(self.surface, self.colour, self.rect)

Then in the client I made a new function called startMenu() that would be run before the main game loop. This start menu is a basic white background with the name of the game displayed in the middle and three buttons.

def startMenu():  
 intro = True  
  
 while intro:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
 # displays text on the screen and generates three buttons  
 gameDisplay.fill((255, 255, 255))  
 button1 = Button(main, gameDisplay, 50, 350, 100, 50, green)  
 button2 = Button(quit, gameDisplay, 200, 350, 100, 50, green)

button3 = Button((os.system(Server.py)), gameDisplay, 350, 350, 100, 50, green)  
  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects("Network Jump", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
 # checks to see if the button has been pressed  
 button1.mouseCheck()  
 button2.mouseCheck()

button3.mouseCheck()  
  
 pygame.display.update()

Using <https://stackoverflow.com/questions/7974849/how-can-i-make-one-python-file-run-another> I tried to find a way to get the button to run the server file concurrently while still using the client file, this way the server could be launched from the client, however using the method I found here lead to the client acting as the server. This meant that I could not run the server from the client. So instead, I reverted to having two buttons and just running the server separately.

def startMenu():  
 intro = True  
  
 while intro:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
 # displays text on the screen and generates three buttons  
 gameDisplay.fill((255, 255, 255))  
 button1 = Button(main, gameDisplay, 100, 350, 100, 50, green)  
 button2 = Button(quit, gameDisplay, 300, 350, 100, 50, green)  
  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects("Network Jump", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
 # checks to see if the button has been pressed  
 button1.mouseCheck()  
 button2.mouseCheck()  
  
 pygame.display.update()

This change made it so I could have a working startmenu and still have the same functionality as in prototype 2.

As for the text, I implemented two other functions (textObjects and messageDisplay)

# used to render the text and set the colour

def textObjects(text, font):  
 textSurface = font.render(text, True, (0, 0, 0))  
 return textSurface, textSurface.get\_rect()

# used to display text in a set position as well as the font type  
def messageDisplay(text):  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects(text, text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
  
 pygame.display.update()

When running the game now, the player is met with the following screen:



Game Title

Quit Button

Start Button

Once the server is started and the player presses the start button this happens:

<https://i.gyazo.com/ac09171abc39924920309aea2faa62bc.gif>

This means the buttons have perfect functionality. However I would like to implement a system where the game waits for both clients to connect before starting game play.

To do this I made a new function similar to the start menu that would act as a waiting area.

def waitForAll():  
 waiting = True  
 connection = ClientConnection()  
 myPlayerPosition = connection.connect()  
  
 while waiting:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
 # displays text on the screen  
 gameDisplay.fill((255, 255, 255))  
 text = pygame.font.Font('freesansbold.ttf', 40)  
 textSurf, textRect = textObjects("Waiting for other player", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
 pygame.display.update()  
 connected = connection.getPlayerCount()  
 # waits for both players to be connected before starting the game  
 if connected == 2:  
 waiting = False  
 main(connection, myPlayerPosition)

But before this would work I had to make some small additions to the clientConnection and server to check whether both clients had connected.

clientConnection change:

# gets the number of players connected to the server  
def getPlayerCount(self):  
 self.client.send(pickle.dumps("getPlayerCount"))  
 return pickle.loads(self.client.recv(2048))

Server changes:

# reply with other player position  
elif data == "getPlayerCount":  
 reply = currentConnections

currentPlayer = 0  
currentConnections = 0  
while True:  
 connection, addr = s.accept()  
 print("Connected to:", addr)  
 start\_new\_thread(threaded\_client, (connection, currentPlayer))  
 currentPlayer += 1  
 currentConnections += 1

Here the server tracks the number of connections based on the number of threaded clients. When the clientConnection sends the message getPlayerCount the server returns the number of threaded clients. And then if the number of threaded clients is 2 the game can continue.

However for any clientconnections to begin before the main game started I had to move where the player first connects to the server. I moved this into the waitForAll function and then passed them through into main when main is ran.

Finally I had to change the buttons function given to run waitForAll rather than main in the startMenu function

button1 = Button(waitForAll, gameDisplay, 100, 350, 100, 50, green)

Now when the game is ran it displays as follows.

<https://i.gyazo.com/87b081e35eb0ff7b5e2c38f8dc902896.gif>

Next I decided I would make the screen scroll, so that when new map is eventually generated it can be seen.

This was quite a simple concept, in the sprite class (as all sprites would need to scroll) I added a new method called scroll()

def scroll(self):  
 self.rect.x -= 1

This function is called by the main game loop near the start of the loop, so that every time the loop resets it scrolls every sprite over.

run = True  
while run:  
 myPlayerPosition = myPlayer.getPos()  
 otherPlayerPosition = connection.send(myPlayerPosition)  
 otherPlayer.setPos(otherPlayerPosition)  
  
 # generates a scrolling effect for the players and platforms to simulate the screen scrolling to the right  
 myPlayer.scroll()  
 for platform in platformSpriteList.sprites():  
 platform.scroll()

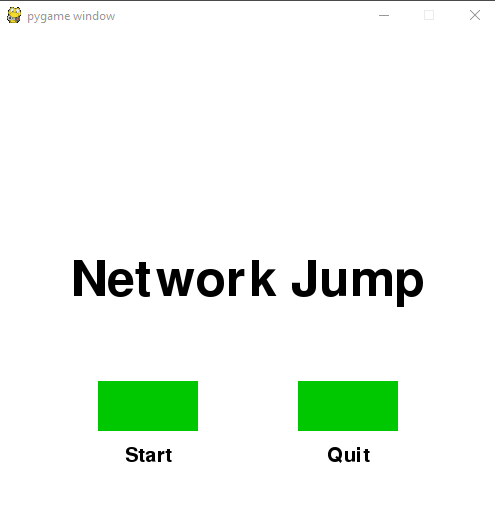
<https://i.gyazo.com/0717e87c0ec2e41b558d4b61093f13a4.gif>

This clip shows that the screen appears to scroll to the right continuously as I saw this as a simpler fix than implementing an actual camera into python.

Next on the buttons I decided to add text so that people could tell what they are pressing.

To do this In the client file I added a script that would display text just below the buttons

# displays text on the screen and generates three buttons  
gameDisplay.fill((255, 255, 255))  
button1 = Button(waitForAll, gameDisplay, 100, 350, 100, 50, green)  
button2 = Button(quit, gameDisplay, 300, 350, 100, 50, green)  
  
text = pygame.font.Font('freesansbold.ttf', 50)  
textSurf, textRect = textObjects("Network Jump", text)  
textRect.center = ((displayWidth / 2), (displayHeight / 2))  
gameDisplay.blit(textSurf, textRect)



This worked perfectly so I moved on to the next part.

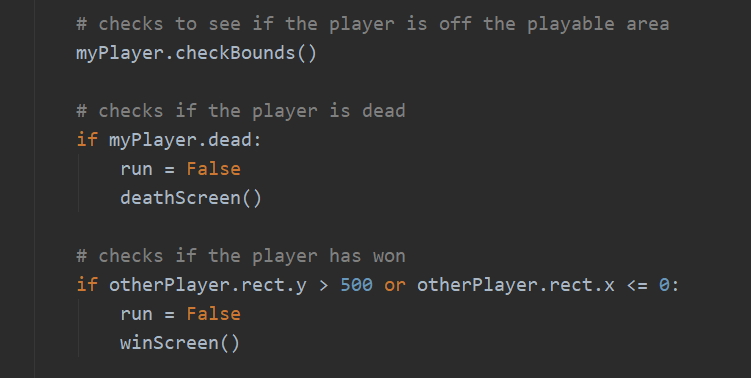
I then made a win and lose screen as well as a way for the game to recognise a winner and a loser.

In the player class I added a new method that checks to make sure the player hasn’t gone outside of the boundaries of the screen.

def checkBounds(self):  
 if self.rect.y >= 520:  
 self.dead = True  
  
 if self.rect.x < 0:  
 self.dead = True  
  
 elif self.rect.x >= 480:  
 self.rect.x = 480

This makes sure that if the player is off either the bottom or the back of the play area the players attribute dead is set to true. This way the client can ask the player if they are dead and determine who won.

Then in the client I added this:



It checks if the clients own player is dead or whether the other clients player is outside the boundaries of the screen and decides a winner from that.

I then had to make a winner and loser screen.

These were largely based on the startmenu so were simple to implement into the game

def winScreen():  
 winning = True  
  
 while winning:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 # displays text on the screen and generates a button  
 gameDisplay.fill((255, 255, 255))  
 button1 = Button(startMenu, gameDisplay, 200, 350, 100, 50, green)  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects("You Won!", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
  
 button1.mouseCheck()  
  
 pygame.display.update()

def deathScreen():  
 dead = True  
  
 while dead:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
 # displays text on the screen and generates a button  
 gameDisplay.fill((255, 255, 255))  
 button1 = Button(startMenu, gameDisplay, 200, 350, 100, 50, green)  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects("You Died", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
  
 button1.mouseCheck()  
  
 pygame.display.update()

These worked perfectly and served their purpose as well as gave the player the option of returning to the startmenu when the game finished.

<https://i.gyazo.com/e5e56ad96eef9b11e20adc476740aabf.gif>

Finally, I Began work on the procedural generation of the map.

First, I changed the spawn area and its generation:

# Initialises Spawn Area  
for x in range(5):  
 startFloor = Wall((100 \* x), 480, 100, 20)  
 platformSpriteList.add(startFloor)  
FirstPlat = Wall(520, 400, 80, 20)  
platformSpriteList.add(FirstPlat)

This made it so that the first platform is always the same so that the procedurally generated platforms from that point on are based off of that original one. But it also meant that there were 6 platforms in the sprite group (platformSpriteList) at all times, so deciding when to generate a new platform can be based off of that.

Next, I made a way for the game to decide when a platform should be removed from the sprite group.

Inside of the scroll handling, I added a check for the platforms, this way it can do both at the same time rather than having to go back through the sprite group again later.

# generates a scrolling effect for the players and platforms to simulate the screen scrolling to the right  
myPlayer.scroll()  
for platform in platformSpriteList.sprites():  
 platform.scroll()  
  
 # if the platform is off the back of the screen it removes it from the sprite group,  
 # which prevents it being redrawn  
 if (platform.rect.x + platform.rect.width) < 0:  
 platform.kill()

Finally, to do the procedural generation I originally intended on generating the map server side, however with the way I was using threads the generation would be different for both clients. Instead, I told clients which Player number they were. Then dependant on this it would decide how they receive the map.

On the server, inside of the threaded client (where it knows which player is which):

if player == 0:  
 playerPositions[0] = data  
 reply = (playerPositions[1], 0)  
else:  
 playerPositions[1] = data  
 reply = (playerPositions[0], 1)

Then on the client I had to change how it handles the initial connection and data exchange.

run = True  
while run:  
 # gets the position data of the other player in exchange for their own position. Also gives client its number  
 myPlayerPosition = myPlayer.getPos()  
 connectionData = connection.send(myPlayerPosition)  
 otherPlayerPosition = connectionData[0]  
 playerNumber = connectionData[1]  
 otherPlayer.setPos(otherPlayerPosition)

I also had to then code the way the client gets its new platforms. These were to be generated on the first client, player 0 and sent to the server, the other client would then ask for this data back from the server.

if playerNumber == 0:  
 if len(platformSpriteList) < 6:  
 # generates a new platform based on the previous  
 lastPlatY = platformSpriteList.sprites()[4].rect.y  
 newPlatY = lastPlatY + (random.choice([-1, 1]) \* 80)  
 if newPlatY == 560:  
 newPlatY = 400  
 if newPlatY == 0:  
 newPlatY = 160  
 connection.sendPlat(newPlatY)  
  
 # adds new platform to the sprite group to be drawn  
 newPlatform = Wall(520, newPlatY, 80, 20)  
 platformSpriteList.add(newPlatform)  
  
else:  
 # gets the position of the new platform generated by the other client  
 if len(platformSpriteList) < 6:  
 newPlatY = connection.getPlat()  
  
 # adds new platform to the sprite group to be drawn  
 newPlatform = Wall(520, newPlatY, 80, 20)  
 platformSpriteList.add(newPlatform)

And lastly I had to make a way for the server to take in this data and return it to the other client, so in client Connection I made two new methods.

# gives the position of the new platform to the server  
def sendPlat(self, platY):  
 self.client.send(pickle.dumps((platY, "sendPlat")))  
  
# gets the position of the new platform from the server  
def getPlat(self):  
 self.client.send(pickle.dumps("newPlat"))  
 return pickle.loads(self.client.recv(2048))

In addition, in the server a way to handle these requests:

try:  
 if data[1] == "sendPlat":  
 newPlatY = data[0]  
except:  
 pass

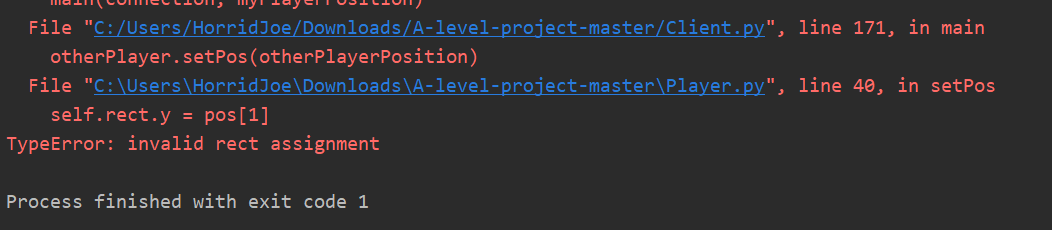
# sends the new platform's y position to the client  
elif data == "newPlat":  
 reply = newPlatY[0]

This should allow the clients to receive the same platform data as each other.

However during testing it created a new problem, one client would crash.

[**https://i.gyazo.com/b4f451b544cf13128d2e50457d4eb6f1.gif**](https://i.gyazo.com/b4f451b544cf13128d2e50457d4eb6f1.gif)

While this did cause a new error for the client, it also showed that the platform generation worked perfectly.



To fix this problem I first put a print statement before and after the connection.getPlat in the client file to see where it breaks as the error given did not seem to be of much help.

if playerNumber == 1:  
 # gets the position of the new platform generated by the other client  
 if len(platformSpriteList) < 6:

print("test 1")  
 newPlatY = connection.getPlat()

print("test 2")

When running the client now, the crashing client (client number 2) would only print test 1 meaning it never reaches the second print statement.

I then realised in the server file getPlat() was causing the client to freeze, as the new platforms y position wasn’t being returned.

This meant the problem was originating in the server. The try and except block was the only possible cause for this so to resolve the problem I removed it, and left what the block enclosed. Upon doing so, the code ran as intended.

try:  
 if data[1] == "sendPlat":  
 newPlatY[0] = data[0]  
except:  
 pass

# sets platform position from client 1's generation algorithm  
elif data[1] == "sendPlat":  
 newPlatY[0] = data[0]

To test my fix I disabled the gravity attribute, this helped me to be able to play both clients.

<https://i.gyazo.com/d6e736bf3f60c59425fe9fb037f4ca05.gif>

During this testing I realised it would be much more helpful to the user if the buttons were labelled so I added a quick fix for this. Inside the button file, I added a new method that would draw text on top of the middle of the box. The text is defined in the generation of the button.

def buttonText(self):  
 text = pygame.font.Font('freesansbold.ttf', 30)  
 textSurf, textRect = textObjects(self.text, text)  
 textRect.center = ((self.rect.x + 50), (self.rect.y + 25))  
 self.surface.blit(textSurf, textRect)

Inside the mousecheck function, I would then call this function so that it is drawn on top of the button.

if self.rect.x + self.rect.width > mouse[0] > self.rect.x and\  
 self.rect.y + self.rect.height > mouse[1] > self.rect.y:  
 pygame.draw.rect(self.surface, bright\_green, self.rect)  
 self.buttonText()  
  
 if pygame.mouse.get\_pressed()[0]:  
 self.function()  
  
else:  
 pygame.draw.rect(self.surface, self.colour, self.rect)  
 self.buttonText()

This produced a perfect result:



**Final Code:**

**Client File**

import pygame  
from ClientConnection import ClientConnection  
from Player import Player  
from Wall import Wall  
from Button import Button  
import random  
  
  
# Create game display  
displayWidth = 500  
displayHeight = 500  
gameDisplay = pygame.display.set\_mode((displayWidth, displayHeight))  
pygame.display.set\_caption("Client")  
platformSpriteList = pygame.sprite.Group()  
playerSpriteList = pygame.sprite.Group()  
  
# Define colours  
  
red = (200, 0, 0)  
green = (0, 200, 0)  
  
bright\_red = (255, 0, 0)  
bright\_green = (0, 255, 0)  
  
pygame.font.init()  
  
  
def textObjects(text, font):  
 textSurface = font.render(text, True, (0, 0, 0))  
 return textSurface, textSurface.get\_rect()  
  
  
# used to display text  
def messageDisplay(text):  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects(text, text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
  
 pygame.display.update()  
  
  
# quits the game when run  
def quit():  
 pygame.quit()  
 exit()  
  
  
def startMenu():  
 intro = True  
  
 while intro:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
 # displays text on the screen and generates three buttons  
 gameDisplay.fill((255, 255, 255))  
 button1 = Button(waitForAll, gameDisplay, 100, 350, 100, 50, green, "start")  
 button2 = Button(quit, gameDisplay, 300, 350, 100, 50, green, "quit")  
  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects("Network Jump", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
 # checks to see if the button has been pressed  
 button1.mouseCheck()  
 button2.mouseCheck()  
  
 pygame.display.update()

def waitForAll():  
 waiting = True  
 connection = ClientConnection()  
 myPlayerPosition = connection.connect()  
  
 while waiting:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
 # displays text on the screen  
 gameDisplay.fill((255, 255, 255))  
 text = pygame.font.Font('freesansbold.ttf', 40)  
 textSurf, textRect = textObjects("Waiting for other player", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
 pygame.display.update()  
 connected = connection.getPlayerCount()  
 # waits for both players to be connected before starting the game  
 if connected == 2:  
 waiting = False  
 main(connection, myPlayerPosition)  
  
  
def deathScreen():  
 dead = True  
  
 while dead:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
 # displays text on the screen and generates a button  
 gameDisplay.fill((255, 255, 255))  
 button1 = Button(startMenu, gameDisplay, 200, 350, 100, 50, green)  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects("You Died", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
  
 button1.mouseCheck()  
  
 pygame.display.update()  
  
  
def winScreen():  
 winning = True  
  
 while winning:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 # displays text on the screen and generates a button  
 gameDisplay.fill((255, 255, 255))  
 button1 = Button(startMenu, gameDisplay, 200, 350, 100, 50, green)  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects("You Won!", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
  
 button1.mouseCheck()  
  
 pygame.display.update()  
  
  
# Main Program  
def main(connection, myPlayerPosition):  
 # Upon connection receives player position from server.  
 otherPlayerPosition = (0, 0)  
  
 pygame.init()  
  
 # Initialises player sprites  
 myPlayer = Player(myPlayerPosition, 20, 20, (255, 0, 0))  
 otherPlayer = Player(otherPlayerPosition, 20, 20, (0, 0, 255))  
  
 # Initialises Spawn Area  
 for x in range(5):  
 startFloor = Wall((100 \* x), 480, 100, 20)  
 platformSpriteList.add(startFloor)  
 FirstPlat = Wall(520, 400, 80, 20)  
 platformSpriteList.add(FirstPlat)  
  
 # Adds player characters to sprite list  
 playerSpriteList.add(otherPlayer)  
 playerSpriteList.add(myPlayer)  
  
 clock = pygame.time.Clock()  
  
 run = True  
 while run:  
 # gets the position data of the other player in exchange for their own position. Also gives client its number  
 myPlayerPosition = myPlayer.getPos()  
 connectionData = connection.send(myPlayerPosition)  
 otherPlayerPosition = connectionData[0]  
 playerNumber = connectionData[1]  
 otherPlayer.setPos(otherPlayerPosition)  
  
 # generates a scrolling effect for the players and platforms to simulate the screen scrolling to the right  
 myPlayer.scroll()  
 for platform in platformSpriteList.sprites():  
 platform.scroll()  
  
 # if the platform is off the back of the screen it removes it from the sprite group,  
 # which prevents it being redrawn  
 if (platform.rect.x + platform.rect.width) < 0:  
 platform.kill()  
  
 if playerNumber == 0:  
 if len(platformSpriteList) < 6:  
 # generates a new platform based on the previous  
 lastPlatY = platformSpriteList.sprites()[4].rect.y  
 newPlatY = lastPlatY + (random.choice([-1, 1]) \* 80)  
 if newPlatY == 560:  
 newPlatY = 400  
 if newPlatY == 0:  
 newPlatY = 160  
 connection.sendPlat(newPlatY)  
  
 # adds new platform to the sprite group to be drawn  
 newPlatform = Wall(520, newPlatY, 80, 20)  
 platformSpriteList.add(newPlatform)  
  
 else:  
 # gets the position of the new platform generated by the other client  
 if len(platformSpriteList) < 6:  
 newPlatY = connection.getPlat()  
  
 # adds new platform to the sprite group to be drawn  
 newPlatform = Wall(520, newPlatY, 80, 20)  
 platformSpriteList.add(newPlatform)  
  
 # updates the players and platforms  
 platformSpriteList.update()  
 playerSpriteList.update()  
  
 # applies gravity to player character if the player character is not already colliding with a platform  
 if pygame.sprite.spritecollideany(myPlayer, platformSpriteList) is None and myPlayer.dead is False:  
 myPlayer.notCollide()  
 myPlayer.checkBounds()  
 if myPlayer.dead:  
 run = False  
 deathScreen()  
  
 elif pygame.sprite.spritecollideany(myPlayer, platformSpriteList) is not None:  
 myPlayer.falling = False  
  
 # checks to see if the player is off the playable area  
 myPlayer.checkBounds()  
  
 # checks if the player is dead  
 if myPlayer.dead:  
 run = False  
 deathScreen()  
  
 # checks if the player has won  
 if otherPlayer.rect.y > 500 or otherPlayer.rect.x < 0:  
 run = False  
 winScreen()  
  
  
 # Handles exiting game  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 run = False  
  
 myPlayer.move()  
 for i in range(5):  
  
 playerSpriteList.update()  
 playerSpriteList.draw(gameDisplay)  
 myPlayer.jump()  
  
 gameDisplay.fill((255, 255, 255))  
 platformSpriteList.draw(gameDisplay)  
 playerSpriteList.draw(gameDisplay)  
  
 pygame.display.update()  
  
 # sets max fps  
 clock.tick(60)  
 pygame.quit()  
 exit()  
  
  
startMenu()

def waitForAll():  
 waiting = True  
 connection = ClientConnection()  
 myPlayerPosition = connection.connect()  
  
 while waiting:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
 # displays text on the screen  
 gameDisplay.fill((255, 255, 255))  
 text = pygame.font.Font('freesansbold.ttf', 40)  
 textSurf, textRect = textObjects("Waiting for other player", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
 pygame.display.update()  
 connected = connection.getPlayerCount()  
 # waits for both players to be connected before starting the game  
 if connected == 2:  
 waiting = False  
 main(connection, myPlayerPosition)  
  
  
def deathScreen():  
 dead = True  
  
 while dead:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
 # displays text on the screen and generates a button  
 gameDisplay.fill((255, 255, 255))  
 button1 = Button(startMenu, gameDisplay, 200, 350, 100, 50, green, "menu")  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects("You Died", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
  
 button1.mouseCheck()  
  
 pygame.display.update()  
  
  
def winScreen():  
 winning = True  
  
 while winning:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 pygame.quit()  
 quit()  
  
 # displays text on the screen and generates a button  
 gameDisplay.fill((255, 255, 255))  
 button1 = Button(startMenu, gameDisplay, 200, 350, 100, 50, green, "menu")  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects("You Won!", text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
  
 button1.mouseCheck()  
  
 pygame.display.update()

# Main Program  
def main(connection, myPlayerPosition):  
 # Upon connection receives player position from server.  
 otherPlayerPosition = (0, 0)  
  
 pygame.init()  
  
 # Initialises player sprites  
 myPlayer = Player(myPlayerPosition, 20, 20, (255, 0, 0))  
 otherPlayer = Player(otherPlayerPosition, 20, 20, (0, 0, 255))  
  
 # Initialises Spawn Area  
 for x in range(5):  
 startFloor = Wall((100 \* x), 480, 100, 20)  
 platformSpriteList.add(startFloor)  
 FirstPlat = Wall(520, 400, 80, 20)  
 platformSpriteList.add(FirstPlat)  
  
 # Adds player characters to sprite list  
 playerSpriteList.add(otherPlayer)  
 playerSpriteList.add(myPlayer)  
  
 clock = pygame.time.Clock()  
  
 run = True  
 while run:  
 # gets the position data of the other player in exchange for their own position. Also gives client its number  
 myPlayerPosition = myPlayer.getPos()  
 connectionData = connection.send(myPlayerPosition)  
 otherPlayerPosition = connectionData[0]  
 playerNumber = connectionData[1]  
 otherPlayer.setPos(otherPlayerPosition)  
  
 # generates a scrolling effect for the players and platforms to simulate the screen scrolling to the right  
 myPlayer.scroll()  
 for platform in platformSpriteList.sprites():  
 platform.scroll()  
  
 # if the platform is off the back of the screen it removes it from the sprite group,  
 # which prevents it being redrawn  
 if (platform.rect.x + platform.rect.width) < 0:  
 platform.kill()  
  
 if playerNumber == 0:  
 if len(platformSpriteList) < 6:  
 # generates a new platform based on the previous  
 lastPlatY = platformSpriteList.sprites()[4].rect.y  
 newPlatY = lastPlatY + (random.choice([-1, 1]) \* 80)  
 if newPlatY == 560:  
 newPlatY = 400  
 if newPlatY == 0:  
 newPlatY = 160  
 connection.sendPlat(newPlatY)  
  
 # adds new platform to the sprite group to be drawn  
 newPlatform = Wall(520, newPlatY, 80, 20)  
 platformSpriteList.add(newPlatform)  
  
 else:  
 # gets the position of the new platform generated by the other client  
 if len(platformSpriteList) < 6:  
 newPlatY = connection.getPlat()  
  
 # adds new platform to the sprite group to be drawn  
 newPlatform = Wall(520, newPlatY, 80, 20)  
 platformSpriteList.add(newPlatform)  
  
 # updates the players and platforms  
 platformSpriteList.update()  
 playerSpriteList.update()  
  
 # applies gravity to player character if the player character is not already colliding with a platform  
 if pygame.sprite.spritecollideany(myPlayer, platformSpriteList) is None and myPlayer.dead is False:  
 myPlayer.notCollide()  
 myPlayer.checkBounds()  
 if myPlayer.dead:  
 run = False  
 deathScreen()  
  
 elif pygame.sprite.spritecollideany(myPlayer, platformSpriteList) is not None:  
 myPlayer.falling = False  
  
 # checks to see if the player is off the playable area  
 myPlayer.checkBounds()  
  
 # checks if the player is dead  
 if myPlayer.dead:  
 run = False  
 deathScreen()  
  
 # checks if the player has won  
 if otherPlayer.rect.y > 500 or otherPlayer.rect.x < 0:  
 run = False  
 winScreen()  
  
  
 # Handles exiting game  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 run = False  
  
 myPlayer.move()  
 for i in range(5):  
  
 playerSpriteList.update()  
 playerSpriteList.draw(gameDisplay)  
 myPlayer.jump()  
  
 gameDisplay.fill((255, 255, 255))  
 platformSpriteList.draw(gameDisplay)  
 playerSpriteList.draw(gameDisplay)  
  
 pygame.display.update()  
  
 # sets max fps  
 clock.tick(60)  
 pygame.quit()  
 exit()  
  
  
startMenu()

# Main Program  
def main(connection, myPlayerPosition):  
 # Upon connection receives player position from server.  
 otherPlayerPosition = (0, 0)  
  
 pygame.init()  
  
 # Initialises player sprites  
 myPlayer = Player(myPlayerPosition, 20, 20, (255, 0, 0))  
 otherPlayer = Player(otherPlayerPosition, 20, 20, (0, 0, 255))  
  
 # Initialises Spawn Area  
 for x in range(5):  
 startFloor = Wall((100 \* x), 480, 100, 20)  
 platformSpriteList.add(startFloor)  
 FirstPlat = Wall(520, 400, 80, 20)  
 platformSpriteList.add(FirstPlat)  
  
 # Adds player characters to sprite list  
 playerSpriteList.add(otherPlayer)  
 playerSpriteList.add(myPlayer)  
  
 clock = pygame.time.Clock()  
  
 run = True  
 while run:  
 # gets the position data of the other player in exchange for their own position. Also gives client its number  
 myPlayerPosition = myPlayer.getPos()  
 connectionData = connection.send(myPlayerPosition)  
 otherPlayerPosition = connectionData[0]  
 playerNumber = connectionData[1]  
 otherPlayer.setPos(otherPlayerPosition)  
  
 # generates a scrolling effect for the players and platforms to simulate the screen scrolling to the right  
 myPlayer.scroll()  
 for platform in platformSpriteList.sprites():  
 platform.scroll()  
  
 # if the platform is off the back of the screen it removes it from the sprite group,  
 # which prevents it being redrawn  
 if (platform.rect.x + platform.rect.width) < 0:  
 platform.kill()  
  
 if playerNumber == 0:  
 if len(platformSpriteList) < 6:  
 # generates a new platform based on the previous  
 lastPlatY = platformSpriteList.sprites()[4].rect.y  
 newPlatY = lastPlatY + (random.choice([-1, 1]) \* 80)  
 if newPlatY == 560:  
 newPlatY = 400  
 if newPlatY == 0:  
 newPlatY = 160  
 connection.sendPlat(newPlatY)  
  
 # adds new platform to the sprite group to be drawn  
 newPlatform = Wall(520, newPlatY, 80, 20)  
 platformSpriteList.add(newPlatform)  
  
 else:  
 # gets the position of the new platform generated by the other client  
 if len(platformSpriteList) < 6:  
 newPlatY = connection.getPlat()

if len(platformSpriteList) < 6:  
 newPlatY = connection.getPlat()  
 if newPlatY == platformSpriteList.sprites()[4]:  
 newPlatY = connection.getPlat()  
 # adds new platform to the sprite group to be drawn  
 newPlatform = Wall(520, newPlatY, 80, 20)  
 platformSpriteList.add(newPlatform)  
  
 # updates the players and platforms  
 platformSpriteList.update()  
 playerSpriteList.update()

# applies gravity to player character if the player character is not already colliding with a platform  
 if pygame.sprite.spritecollideany(myPlayer, platformSpriteList) is None and myPlayer.dead is False:  
 myPlayer.notCollide()  
 myPlayer.checkBounds()  
 if myPlayer.dead:  
 run = False  
 deathScreen()  
  
 elif pygame.sprite.spritecollideany(myPlayer, platformSpriteList) is not None:  
 myPlayer.falling = False  
  
 # checks to see if the player is off the playable area  
 myPlayer.checkBounds()  
  
 # checks if the player is dead  
 if myPlayer.dead:  
 run = False  
 deathScreen()  
  
 # checks if the player has won  
 if otherPlayer.rect.y > 500 or otherPlayer.rect.x < 0:  
 run = False  
 winScreen()  
  
  
 # Handles exiting game  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 run = False  
  
 myPlayer.move()  
 for i in range(5):  
  
 playerSpriteList.update()  
 playerSpriteList.draw(gameDisplay)  
 myPlayer.jump()  
  
 gameDisplay.fill((255, 255, 255))  
 platformSpriteList.draw(gameDisplay)  
 playerSpriteList.draw(gameDisplay)  
  
 pygame.display.update()  
  
 # sets max fps  
 clock.tick(60)  
 pygame.quit()  
 exit()  
  
  
startMenu()

**Server File**

import socket  
from \_thread import \*  
import pickle  
import random  
  
  
# 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)  
  
# opens up port to allow us to begin to use it to connect multiple clients together. Inside the brackets limits the  
# number of possible connections  
s.listen(2)  
print("Waiting for connection, Server Initiated")  
  
newPlatY = [480]  
  
# player start positions  
playerPositions = [[200, 440], [200, 440]]  
  
# keeps track of any deaths  
dead = False  
  
  
# client = connection, player = player number  
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:  
  
 if data == "getPos":  
 reply = (playerPositions[player])  
  
 elif data == "dead":  
 dead = True  
  
 elif data == "checkdead":  
 reply = dead  
  
 # reply with other player position  
 elif data == "getPlayerCount":  
 reply = currentConnections  
  
 # sets platform position from client 1's generation algorithm  
 elif data[1] == "sendPlat":  
 newPlatY[0] = data[0]  
  
 # sends the new platform's y position to the client  
 elif data == "newPlat":  
 print(newPlatY[0])  
 reply = newPlatY[0]  
  
 else:  
 if player == 0:  
 playerPositions[0] = data  
 reply = (playerPositions[1], 0)  
 else:  
 playerPositions[1] = data  
 reply = (playerPositions[0], 1)  
  
 else:  
 print("Lost Connection")  
 client.close()  
 break  
  
 # sends data back to all open threads  
 client.sendall(pickle.dumps(reply))  
  
  
# keeps track of current number of players  
currentPlayer = 0  
currentConnections = 0  
while True:  
 connection, addr = s.accept()  
 print("Connected to:", addr)  
 start\_new\_thread(threaded\_client, (connection, currentPlayer))  
 currentPlayer += 1  
 currentConnections += 1

# reply with other player position  
 elif data == "getPlayerCount":  
 reply = currentConnections  
  
 # sets platform position from client 1's generation algorithm  
 elif data[1] == "sendPlat":  
 newPlatY[0] = data[0]  
  
 # sends the new platform's y position to the client  
 elif data == "newPlat":  
 print(newPlatY[0])  
 reply = newPlatY[0]

# sets the player positions of the client sending the data  
 else:  
 if player == 0:  
 playerPositions[0] = data  
 reply = (playerPositions[1], 0)  
 else:  
 playerPositions[1] = data  
 reply = (playerPositions[0], 1)  
  
 else:  
 print("Lost Connection")  
 client.close()  
 break  
  
 # sends data back to all open threads  
 client.sendall(pickle.dumps(reply))  
  
  
# keeps track of current number of players  
currentPlayer = 0  
currentConnections = 0  
while True:  
 connection, addr = s.accept()  
 print("Connected to:", addr)  
 start\_new\_thread(threaded\_client, (connection, currentPlayer))  
 currentPlayer += 1  
 currentConnections += 1

**clientConnection File**

import socket  
import pickle  
  
  
class ClientConnection:  
 def \_\_init\_\_(self):  
 self.client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 self.server = "192.168.0.37"  
 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"))  
 return self.getPos()  
  
 # gets the position of the objects  
 def getPos(self):  
 return pickle.loads(self.client.recv(2048))  
  
 # gives the position of the new platform to the server  
 def sendPlat(self, platY):  
 self.client.send(pickle.dumps((platY, "sendPlat")))  
  
 # gets the position of the new platform from the server  
 def getPlat(self):  
 self.client.send(pickle.dumps("newPlat"))  
 return pickle.loads(self.client.recv(2048))  
  
 # gets the number of players connected to the server  
 def getPlayerCount(self):  
 self.client.send(pickle.dumps("getPlayerCount"))  
 return pickle.loads(self.client.recv(2048))  
  
 def dead(self):  
 self.client.send(pickle.dumps("Dead"))  
  
 # sends data to the server and receives data other client data  
 def send(self, data):  
 try:  
 self.client.send(pickle.dumps(data))  
 return pickle.loads(self.client.recv(2048))  
 except socket.error as e:  
 print(e)

**Sprites File**

import pygame  
  
  
# child class of pygame.sprite.Sprite  
class Sprite(pygame.sprite.Sprite):  
 def \_\_init\_\_(self, x, y, width, height, colour):  
 pygame.sprite.Sprite.\_\_init\_\_(self)  
 self.image = pygame.Surface([width, height])  
 self.image.fill(colour)  
 self.rect = self.image.get\_rect()  
 self.rect.x = x  
 self.rect.y = y  
  
 def scroll(self):  
 self.rect.x -= 3

**Wall File**

from Sprites import Sprite  
import pygame  
  
  
class Wall(Sprite):  
 def \_\_init\_\_(self, x, y, width, height):  
 Sprite.\_\_init\_\_(self, x, y, width, height, (0, 0, 0))

**Player File**

import pygame  
from Sprites import Sprite  
  
  
# Construction of character  
class Player(Sprite):  
 def \_\_init\_\_(self, posargs, width, height, colour):  
 self.x = posargs[0]  
 self.y = posargs[1]  
 Sprite.\_\_init\_\_(self, self.x, self.y, width, height, colour)  
 self.vel = 5  
 self.upVel = 30  
 self.jumping = False  
 self.gravity = 5  
 self.falling = False  
 self.dead = False  
  
 def move(self):  
 keys = pygame.key.get\_pressed()  
  
 if keys[pygame.K\_d]:  
 self.rect.x += self.vel  
  
 if keys[pygame.K\_a]:  
 self.rect.x -= self.vel  
  
 def jump(self):  
 keys = pygame.key.get\_pressed()  
  
 if keys[pygame.K\_w]:  
 if self.jumping is False and self.falling is False:  
 self.rect.y -= self.upVel  
  
 def notCollide(self):  
 self.rect.y += self.gravity  
 self.falling = True  
  
 def setPos(self, pos):  
 self.rect.x = pos[0]  
 self.rect.y = pos[1]  
  
 def getPos(self):  
 return [self.rect.x, self.rect.y]  
  
 def checkBounds(self):  
 if self.rect.y >= 520:  
 self.dead = True  
  
 if self.rect.x < 0:  
 self.dead = True  
  
 elif self.rect.x >= 480:  
 self.rect.x = 480

**Button File**

import pygame  
  
displayWidth = 500  
displayHeight = 500  
  
red = (200, 0, 0)  
green = (0, 200, 0)  
  
bright\_red = (255, 0, 0)  
bright\_green = (0, 255, 0)  
  
pygame.font.init()  
  
def textObjects(text, font):  
 textSurface = font.render(text, True, (0, 0, 0))  
 return textSurface, textSurface.get\_rect()  
  
  
# used to display text  
def messageDisplay(text):  
 text = pygame.font.Font('freesansbold.ttf', 50)  
 textSurf, textRect = textObjects(text, text)  
 textRect.center = ((displayWidth / 2), (displayHeight / 2))  
 gameDisplay.blit(textSurf, textRect)  
  
 pygame.display.update()  
  
class Button:  
 def \_\_init\_\_(self, function, surface, posx, posy, width, height, colour, text):  
 self.rect = pygame.Rect(posx, posy, width, height)  
 self.colour = colour  
 self.surface = surface  
 self.function = function  
 self.text = text  
 pygame.draw.rect(self.surface, self.colour, self.rect)  
  
 def mouseCheck(self):  
 mouse = pygame.mouse.get\_pos()  
  
 if self.rect.x + self.rect.width > mouse[0] > self.rect.x and\  
 self.rect.y + self.rect.height > mouse[1] > self.rect.y:  
 pygame.draw.rect(self.surface, bright\_green, self.rect)  
 self.buttonText()  
  
 if pygame.mouse.get\_pressed()[0]:  
 self.function()  
  
 else:  
 pygame.draw.rect(self.surface, self.colour, self.rect)  
 self.buttonText()  
  
 def buttonText(self):  
 text = pygame.font.Font('freesansbold.ttf', 30)  
 textSurf, textRect = textObjects(self.text, text)  
 textRect.center = ((self.rect.x + 50), (self.rect.y + 25))  
 self.surface.blit(textSurf, textRect)

**Stakeholder Feedback:**

**This stakeholder meeting was cancelled due to college being converted to distance learning.**