Project Report

Open Source Technology

By:

Naman Jain

CONTENTS

Multiplayer Shooter

Abstract

The aim of the project is to show working of a client server application with the help of socket programming. Best application to show this is through a massive multiplayer online game which has hundreds of clients interacting simultaneously with the server.

For this project, we have created a server which accepts up to 100 requests from clients connected in a local area network. In this way any computer which has the server-side code can act as a server by sharing it's private IP address and then every player can connect to it.

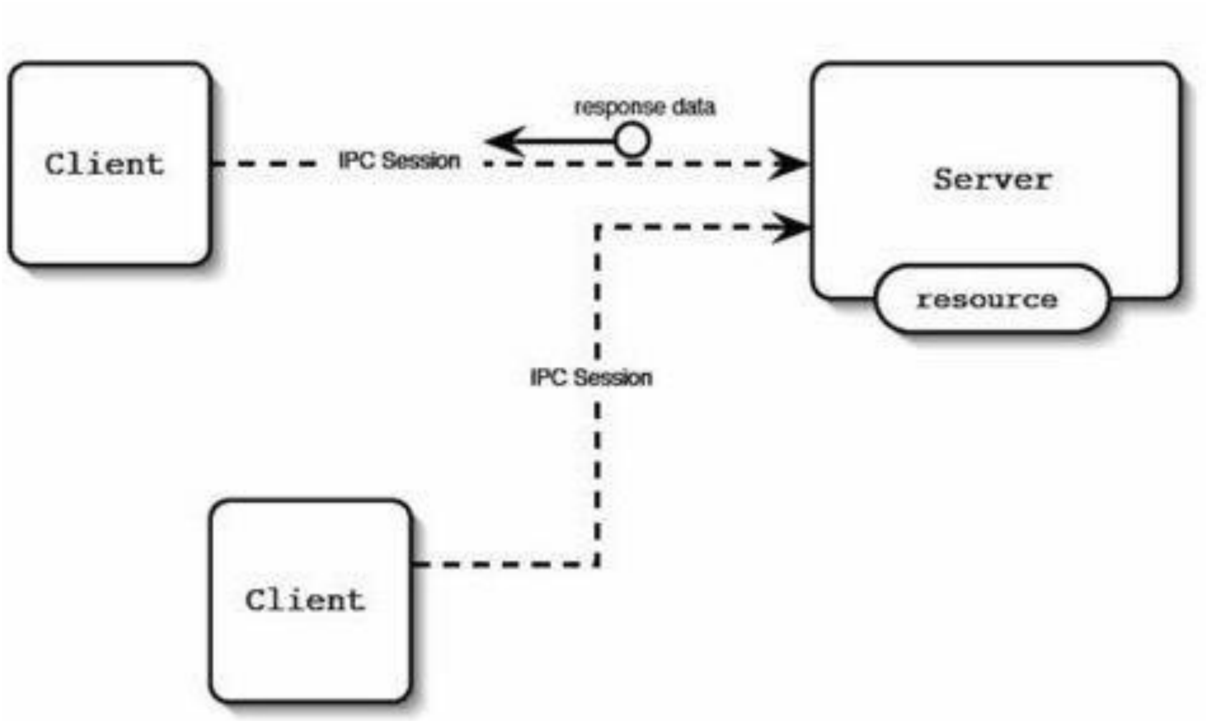
Technologies Used

- **Python :** Used to create the server and client codes in order to facilitate the information flow between the clients/players.

- **Math :** Python library to facilitate the calculation of bullet trajectory.

- **Arcade :** Arcade is an easy-to-learn Python library for creating 2D video games. It is ideal for people learning to program, or developers that want to code a 2D game without learning a complex framework.

How the project works



A server is hosted to facilitate the communication between multiple clients to facilitate the exchange of data such as position of players on the map, bullets shot, spawn points of med-kits.

Every player/client interact with the server and update their position as well as shots they fire which in turn is broadcasted to the other clients to interact with.

How game servers work in today's scenario

- ✓ The most popular multiplayer games like World of Warcraft, Call of Duty or Minecraft, as well as thousands more online games, work basically the same way. Players install a "client" for the game on their computer and then connect to the game servers via an internet connection.

- ✓ These servers are generally distributed around the world as it guarantees access to the game even if one of the login servers is not available. If a server fails, players can still enter the game through the other servers.

- ✓ To ensure fair gaming experience game servers run real time analytics for cheat detection.

- ✓ As the environment is distributed changes in the world received from a client must be propagated to all the others in an "eventually consistent" manner

Problems faced by game servers

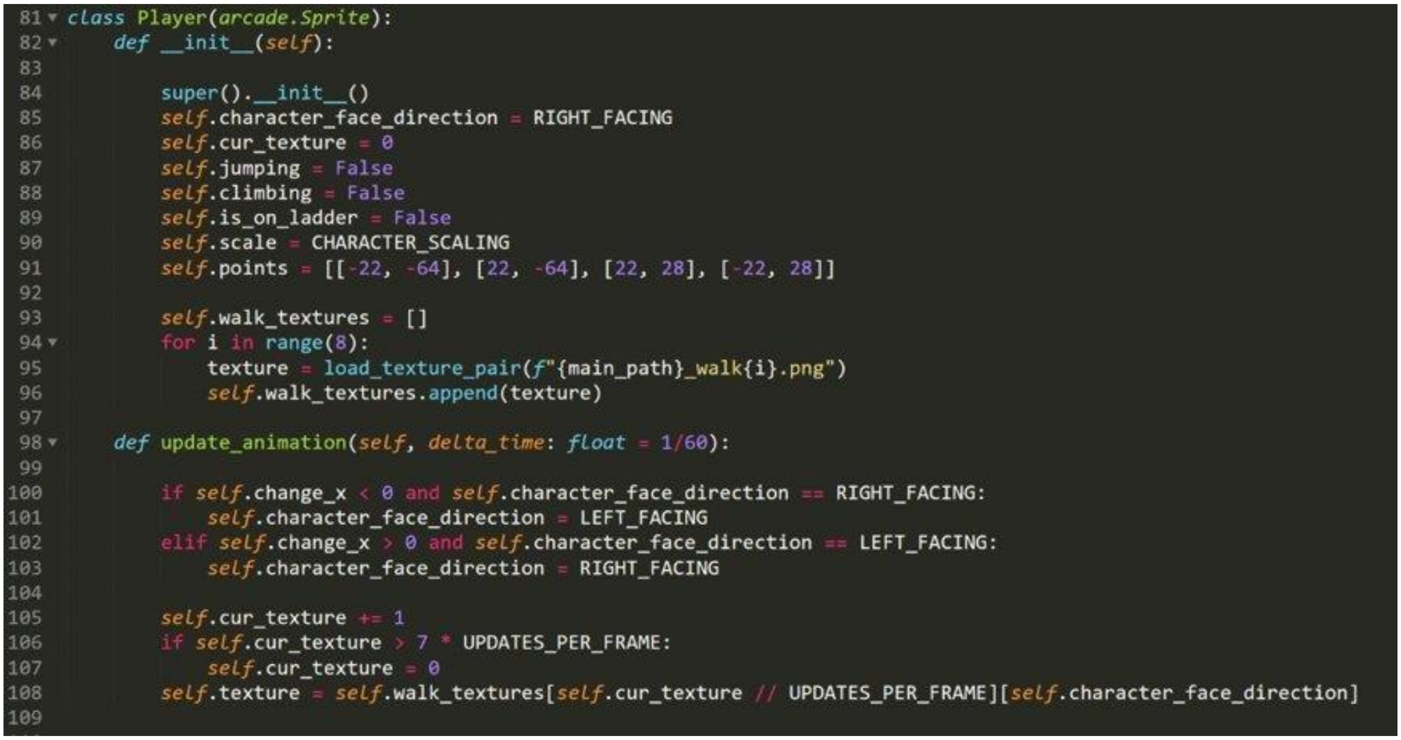
- The most problematic part is the connection between the game server and the client, because we can not influence it. The only thing we can do to get a faster and more efficient connection is to choose a good data center that has connections to a large number of providers. In this way, the connection can be quickly diverted if a provider does not work correctly.

- High latency - or lag - translates to delay in games, because the time it takes for a data packet to reach the server and return to the sender is too large.

- To ensure a good gaming experience for users, most gaming companies invest in deploying infrastructure in data centers specialized in gaming in various parts of the world.

Player animation implementation

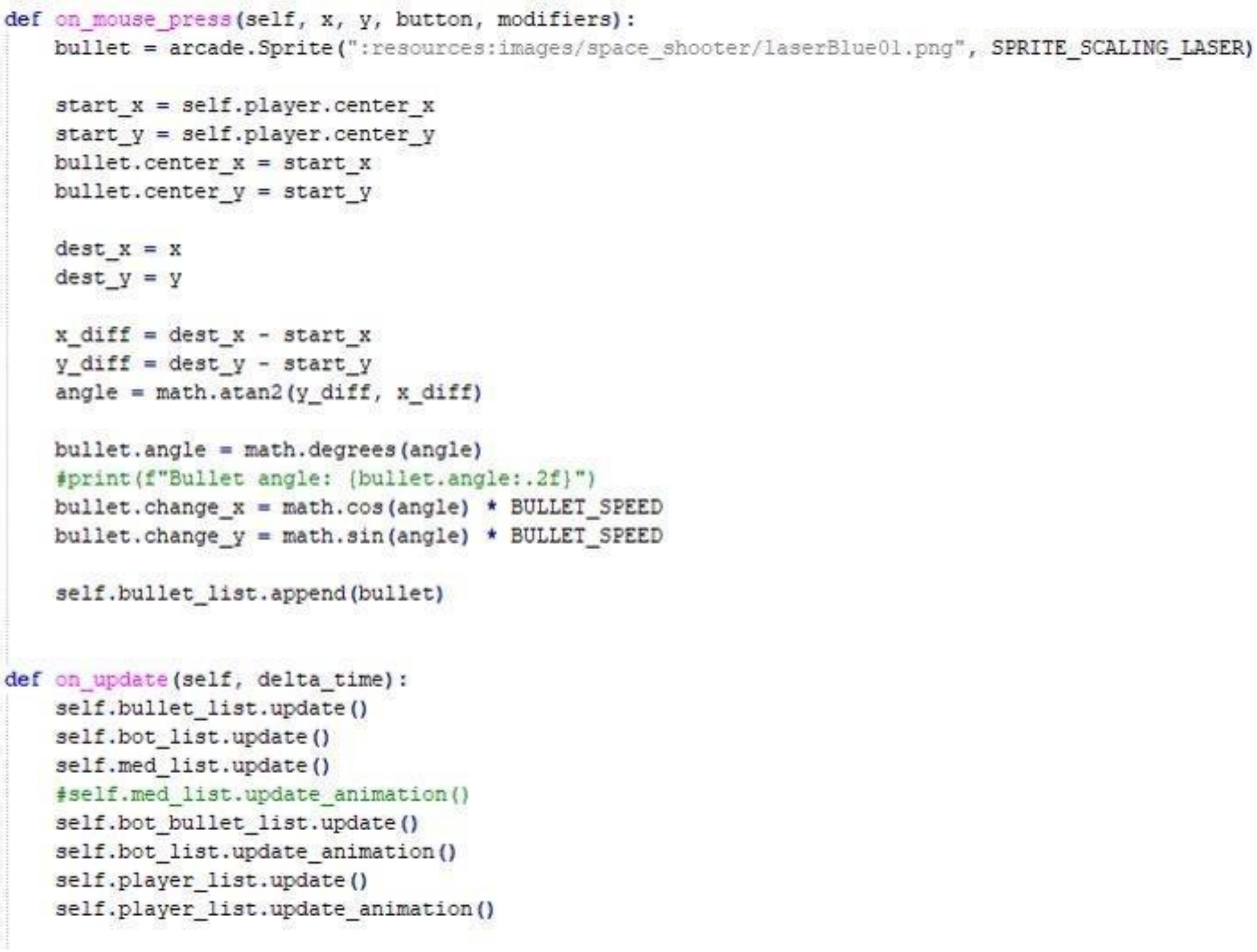
A player class is made with the directory of the stored sprites and a loop through 7 images is run in order to show animations for the movement or idle animation.



```python
class Player(arcade.Sprite):
    def __init__(self):

        super().__init__()
        self.character_face_direction = RIGHT_FACING
        self.cur_texture = 0
        self.jumping = False
        self.climbing = False
        self.is_on_ladder = False
        self.scale = CHARACTER_SCALING
        self.points = [[-22, -64], [22, -64], [22, 28], [-22, 28]]

        self.walk_textures = []
        for i in range(8):
            texture = load_texture_pair(f"{main_path}_walk{i}.png")
            self.walk_textures.append(texture)

    def update_animation(self, delta_time: float = 1/60):

        if self.change_x < 0 and self.character_face_direction == RIGHT_FACING:
            self.character_face_direction = LEFT_FACING
        elif self.change_x > 0 and self.character_face_direction == LEFT_FACING:
            self.character_face_direction = RIGHT_FACING

        self.cur_texture += 1
        if self.cur_texture > 7 * UPDATES_PER_FRAME:
            self.cur_texture = 0
        self.texture = self.walk_textures[self.cur_texture // UPDATES_PER_FRAME][self.character_face_direction]

```

Handling collisions and updating bullet trajectory

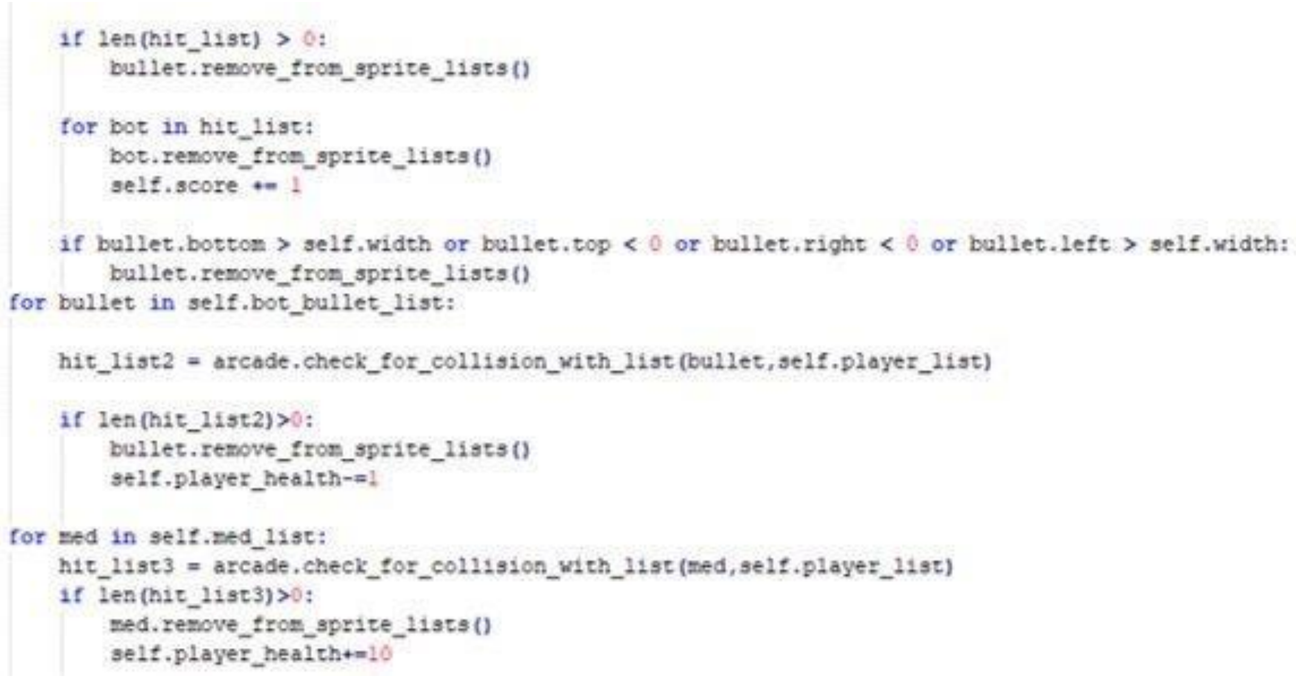
❑Whenever the mouse is pressed, the position of the mouse cursor is noted as well as player's position\



```
def on_mouse_press(self, x, y, button, modifiers):
    bullet = arcade.Sprite(":resources:images/space_shooter/laserBlue01.png", SPRITE_SCALING_LASER)

    start_x = self.player.center_x
    start_y = self.player.center_y
    bullet.center_x = start_x
    bullet.center_y = start_y

    dest_x = x
    dest_y = y

    x_diff = dest_x - start_x
    y_diff = dest_y - start_y
    angle = math.atan2(y_diff, x_diff)

    bullet.angle = math.degrees(angle)
    #print(f"Bullet angle: {bullet.angle:.2f}")
    bullet.change_x = math.cos(angle) * BULLET_SPEED
    bullet.change_y = math.sin(angle) * BULLET_SPEED

    self.bullet_list.append(bullet)


def on_update(self, delta_time):
    self.bullet_list.update()
    self.bot_list.update()
    self.med_list.update()
    #self.med_list.update_animation()
    self.bot_bullet_list.update()
    self.bot_list.update_animation()
    self.player_list.update()
    self.player_list.update_animation()
```

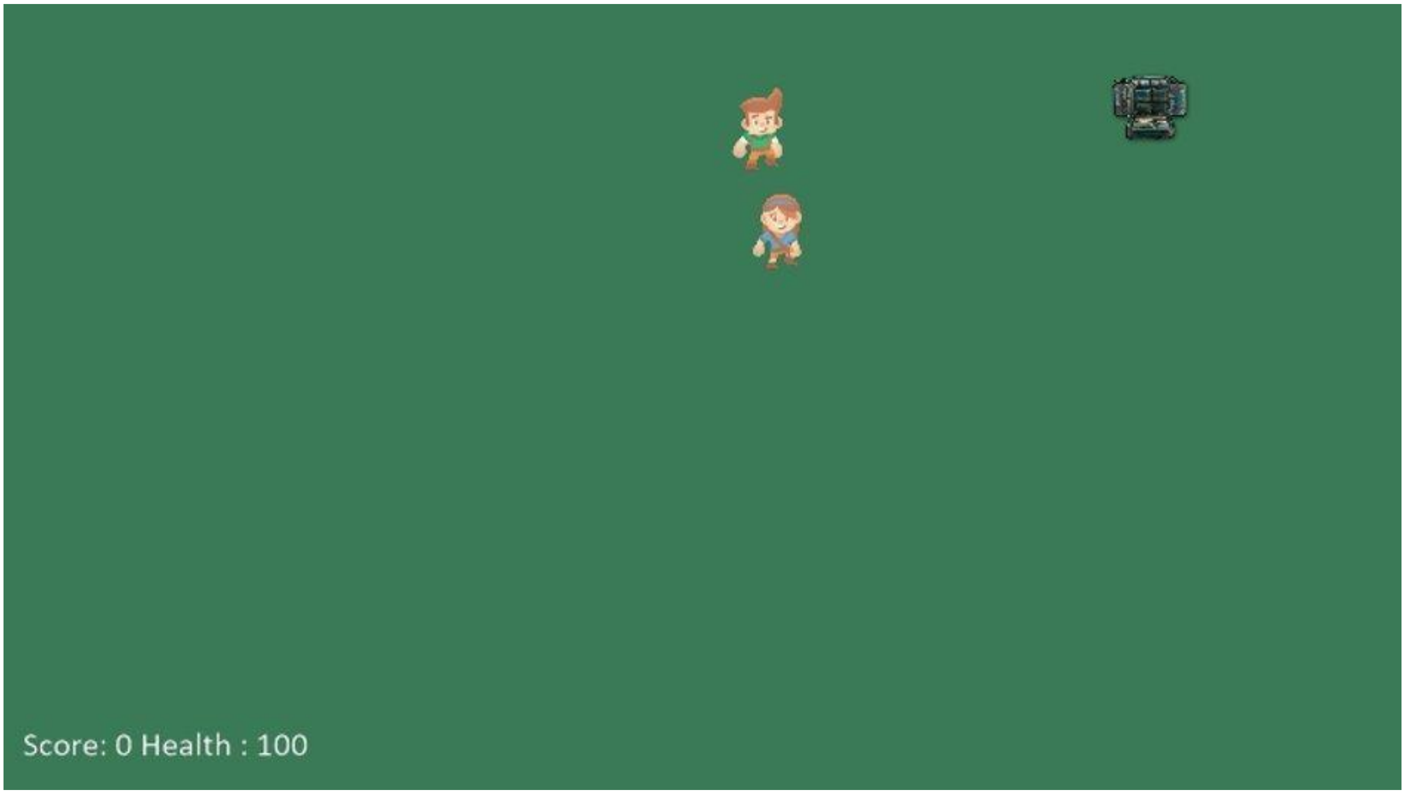
❑ The angle of the shot is calculated with the help of the two positions and a bullet is added to the sprite list.

❑ A collision list is created and every player is checked for collision with the list of bullets and if a collision occurs the bullet is removed from the list and player's health is subtracted accordingly.



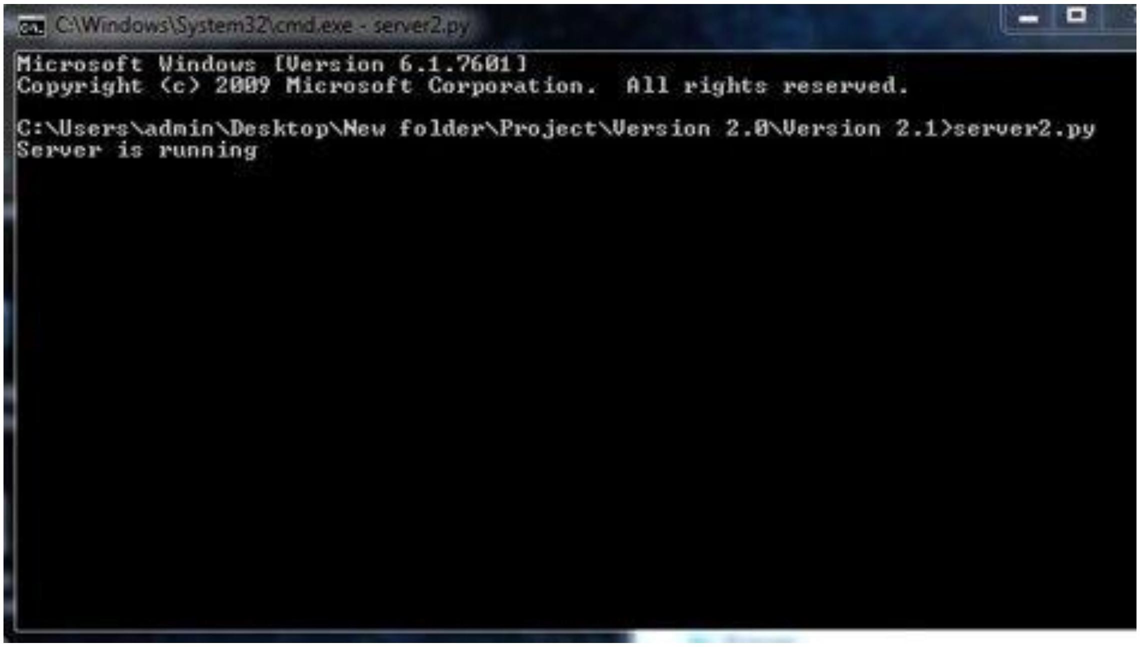
```
    if len(hit_list) > 0:
        bullet.remove_from_sprite_lists()

    for bot in hit_list:
        bot.remove_from_sprite_lists()
        self.score += 1

    if bullet.bottom > self.width or bullet.top < 0 or bullet.right < 0 or bullet.left > self.width:
        bullet.remove_from_sprite_lists()
for bullet in self.bot_bullet_list:

    hit_list2 = arcade.check_for_collision_with_list(bullet,self.player_list)

    if len(hit_list2)>0:
        bullet.remove_from_sprite_lists()
        self.player_health-=1

for med in self.med_list:
    hit_list3 = arcade.check_for_collision_with_list(med,self.player_list)
    if len(hit_list3)>0:
        med.remove_from_sprite_lists()
        self.player_health+=10
```

Demonstration

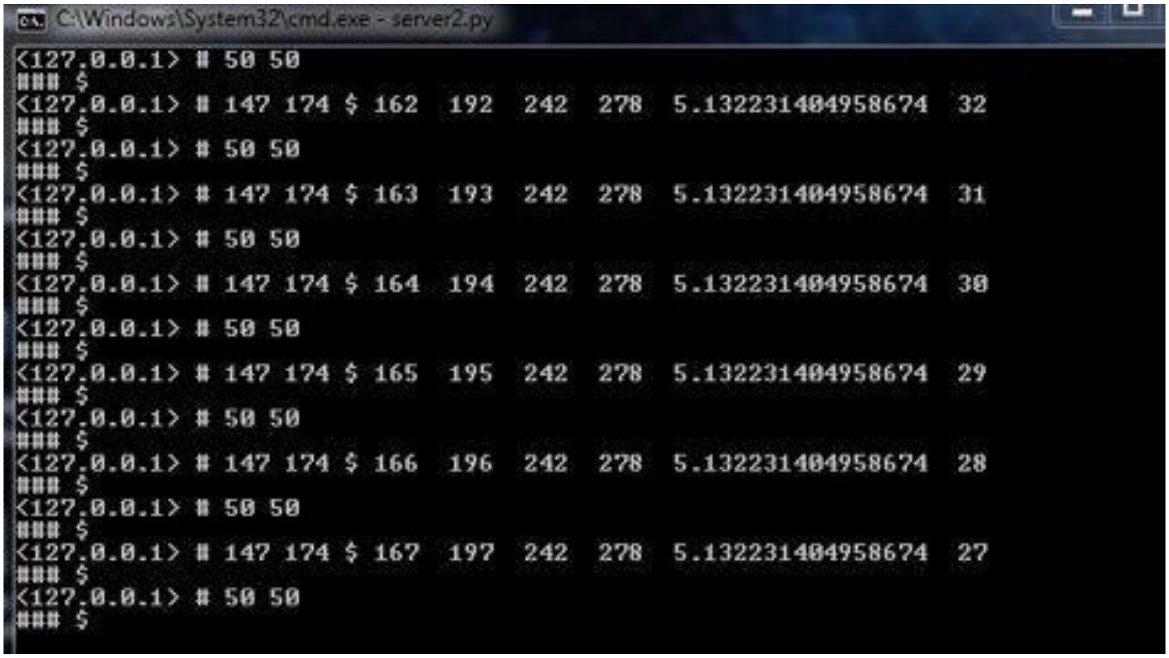


Score: 0 Health : 100

The Server while the game is running

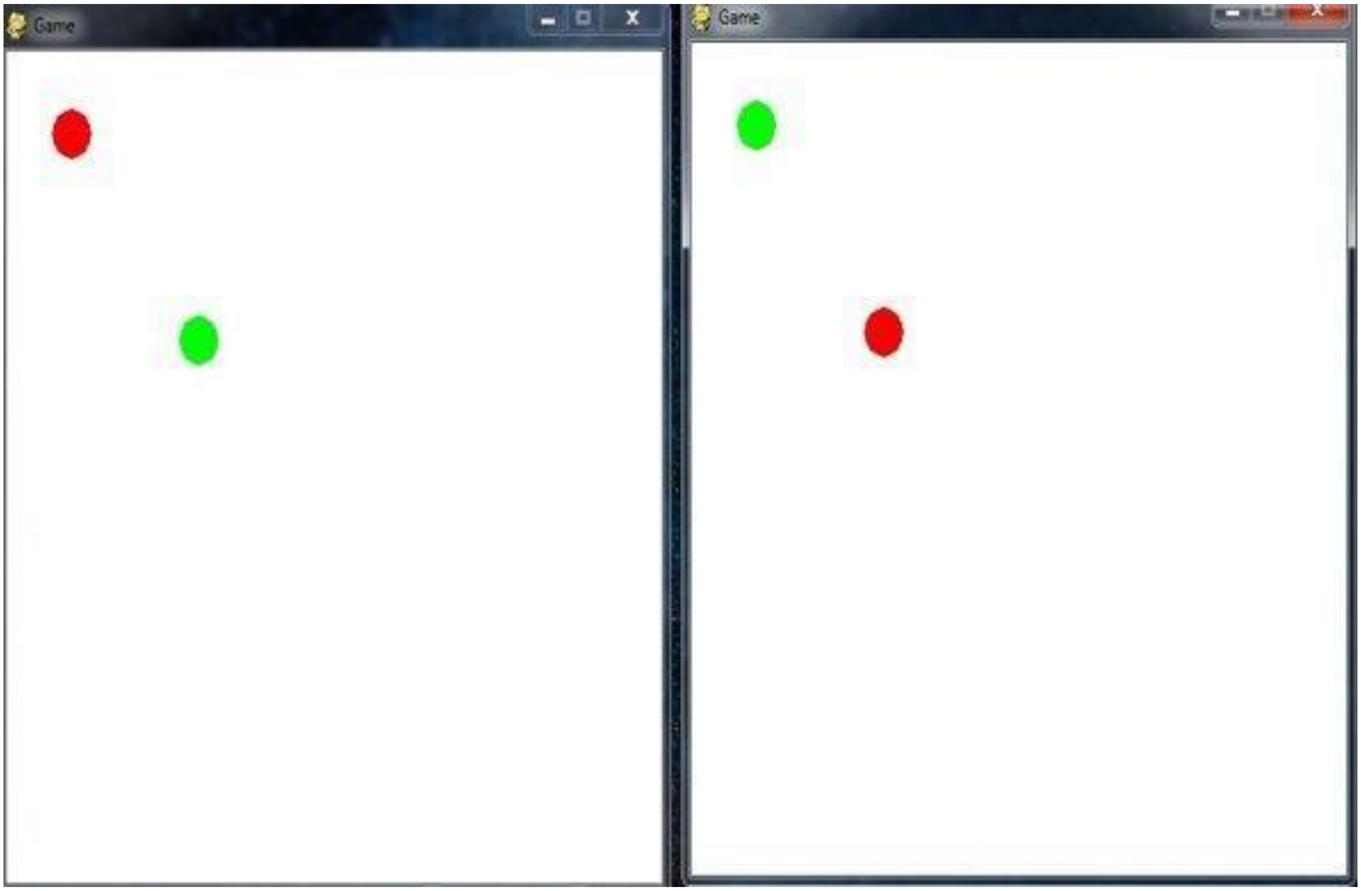


```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\admin\Desktop\New folder\Project\Version 2.0\Version 2.1>server2.py
Server is running
```

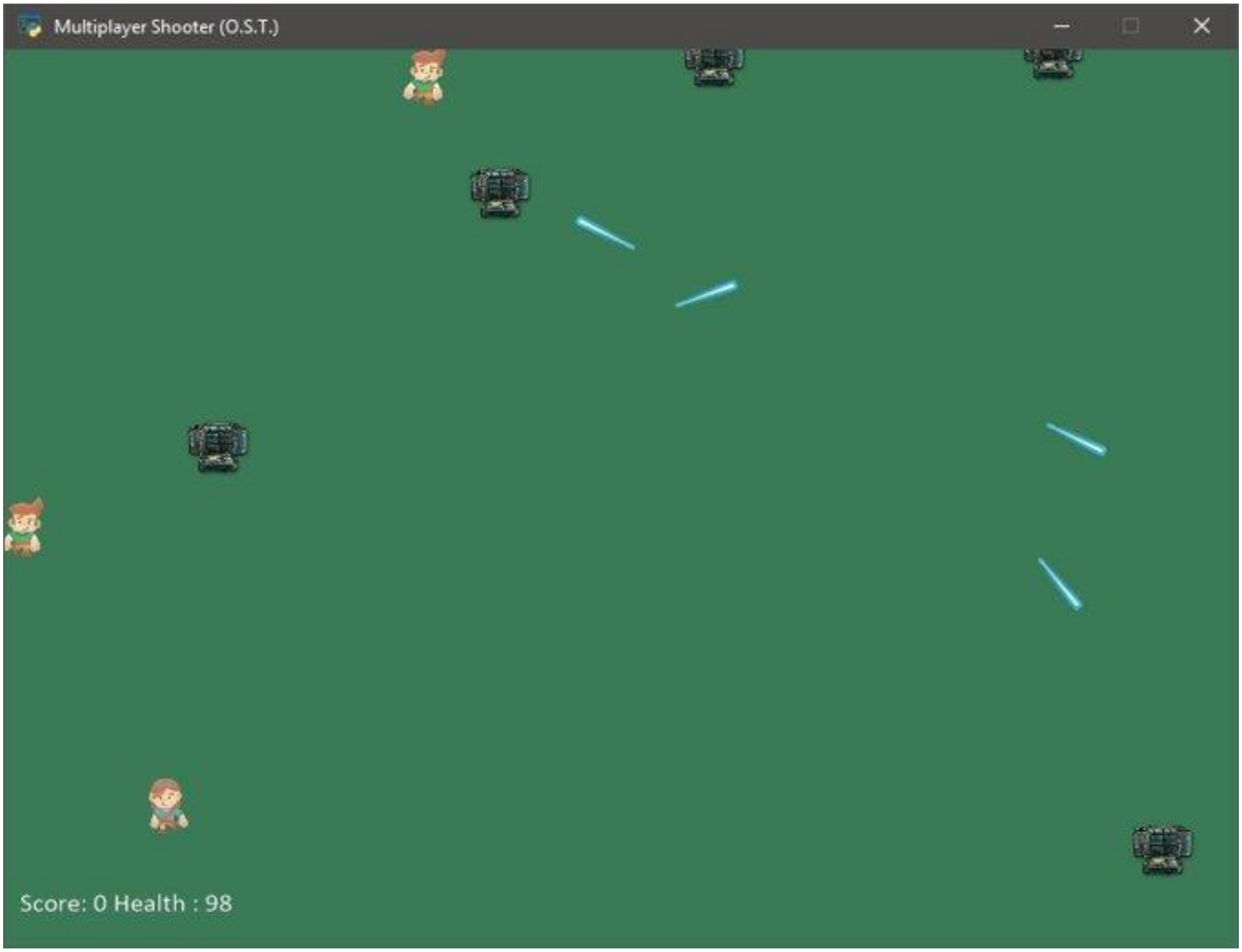


```
(127.0.0.1) # 50 50
### $
(127.0.0.1) # 147 174 $ 162  192  242  278  5.132231404958674  32
### $
(127.0.0.1) # 50 50
### $
(127.0.0.1) # 147 174 $ 163  193  242  278  5.132231404958674  31
### $
(127.0.0.1) # 50 50
### $
(127.0.0.1) # 147 174 $ 164  194  242  278  5.132231404958674  30
### $
(127.0.0.1) # 50 50
### $
(127.0.0.1) # 147 174 $ 165  195  242  278  5.132231404958674  29
### $
(127.0.0.1) # 50 50
### $
(127.0.0.1) # 147 174 $ 166  196  242  278  5.132231404958674  28
### $
(127.0.0.1) # 50 50
### $
(127.0.0.1) # 147 174 $ 167  197  242  278  5.132231404958674  27
### $
(127.0.0.1) # 50 50
### $
```

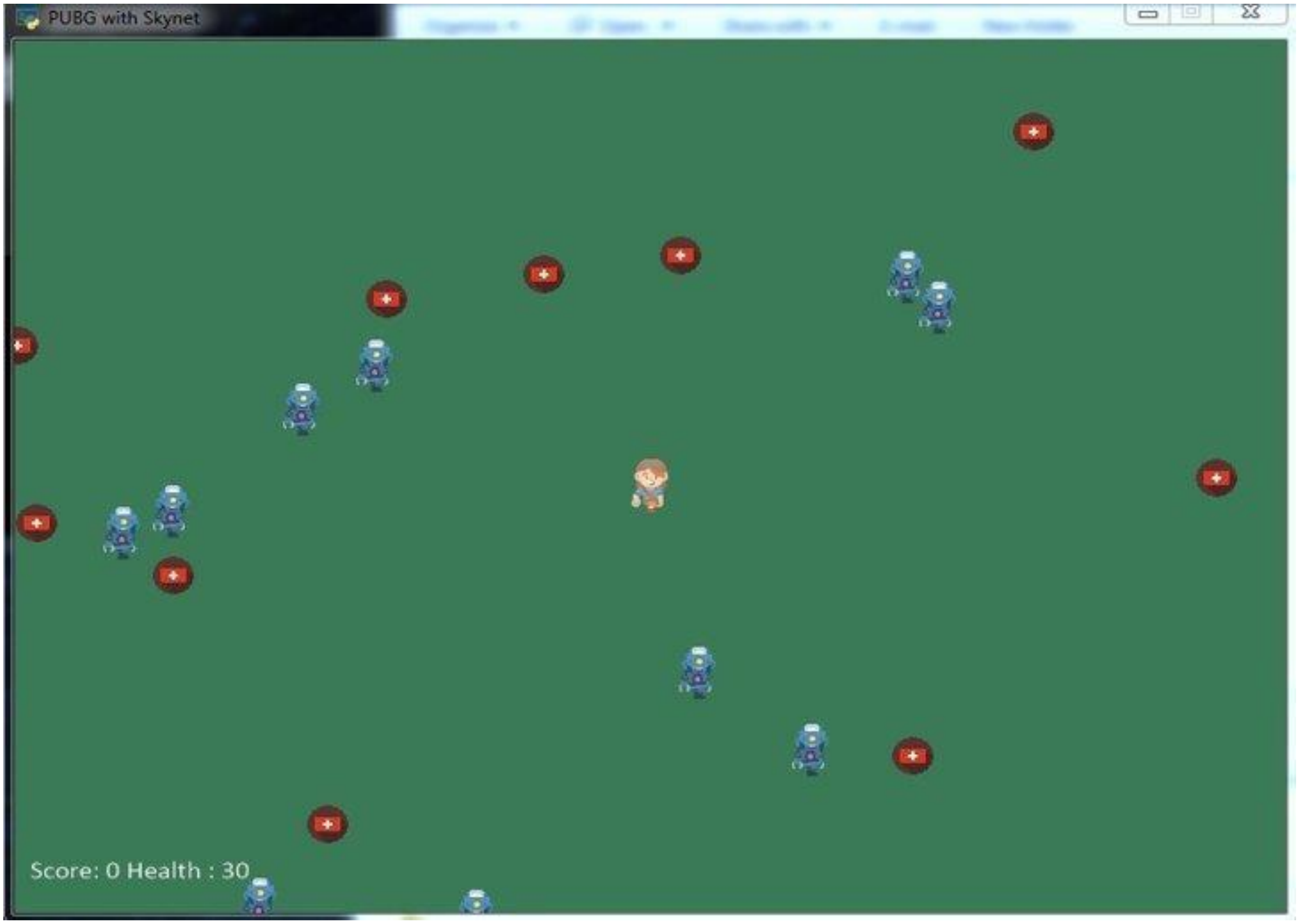
The game in its earlier phase



In the above screenshot, the red dots represent the other players on a client's screen whereas the green ones represent the player playing the game as depicted on the two windows.



A 3 player game with random spawns



An instance of 10 clients playing the game(opponents depicted as the robot sprites) and random medkit spawns.

Reference

https://www.tomlooman.com/multiplayer-shooter-cpp-ue4-preview/

https://arcade-tutorials.readthedocs.io/en/latest/

https://docs.python.org/2/howto/sockets.html

https://arcade-tutorials.readthedocs.io/en/latest/animation/index.html