

TRƯỜNG ĐẠI HỌC SÀI GÒN
KHOA CÔNG NGHỆ THÔNG TIN



PHÁT TRIỂN PHẦN MỀM MÃ NGUỒN MỞ

Lập trình Python với Socket

Lập trình Game Bắn súng URSINA FPS

GVHD: Từ Lãng Phiêu
SV: Nguyễn Minh Tú - 3120410578
Nhữ Văn Tuyền - 3120410597
Nguyễn Đăng Vũ - 3120410622

TP. HỒ CHÍ MINH, NGÀY 20 THÁNG 4 NĂM 2024

Mục lục

1 Lời cảm ơn	2
2 Lời mở đầu	3
3 Phân giới thiệu	4
3.1 Lý do chọn đề tài	4
3.2 Giới thiệu về URSINA	4
3.3 Giới thiệu về Socket	4
3.4 Giới thiệu về chương trình	5
4 Tổng quan chương trình	6
4.1 Game	6
4.1.1 Player	6
4.1.2 Setup	6
4.1.3 Enemy	7
4.1.4 Floor	7
4.1.5 Map	8
4.1.6 Bullet	8
4.1.7 Network	9
4.1.8 Main	10
4.2 Server	11
4.2.1 Main	11
5 Kết luận	13
6 Tài liệu tham khảo	14



1 Lời cảm ơn

Dầu tiên, nhóm chúng em xin gửi lời cảm ơn chân thành đến **Trường Đại học Sài Gòn** đã bố trí sắp xếp môn học **Phát triển phần mềm mã nguồn mở**. Đặc biệt, em xin gửi lời cảm ơn sâu sắc nhất đến Giảng viên ThS. **Từ Lãng Phiêu** đã chỉ bảo và truyền đạt những kiến thức quý báu cho chúng em trong suốt thời gian học tập vừa qua. Trong thời gian tham gia lớp học của thầy, chúng em đã có thêm cho mình nhiều kiến thức bổ ích, tinh thần học tập hiệu quả, nhờ đó có thể tích lũy được vốn kiến thức nền tảng, giúp chúng em rất nhiều trong suốt thời gian xây dựng đồ án. Đây sẽ là hành trang quý báu để chúng em tiếp bước sau này.

Bộ môn **Phát triển phần mềm mã nguồn mở** là một môn học thú vị, vô cùng bổ ích và tính thực tế cao với những kiến thức mới lạ. Cung cấp đầy đủ kiến thức, gắn liền với nhu cầu thực tiễn của người dùng hiện nay. Tuy nhiên, do vốn kiến thức còn hạn chế và kiến thức chưa đủ nên khi làm đồ án có nhiều vấn đề xảy ra. Chúng em đã cố gắng hết khả năng để phát triển đề tài này nhưng chắc sẽ khó tránh khỏi những thiếu sót cơ bản và những chỗ chưa chính xác. Kính mong thầy xem xét và góp ý cho nhóm em về đề tài này để nhóm em có thể hoàn thiện hơn.

Nhóm chúng em xin chân thành cảm ơn!



2 Lời mở đầu

Python là ngôn ngữ lập trình máy tính cấp cao thường được sử dụng để xây dựng trang web và các phần mềm, tự động hóa các tác vụ và tiến hành phân tích dữ liệu. Python thân thiện với người mới bắt đầu, nó đã trở thành một trong những ngôn ngữ lập trình được sử dụng rộng rãi nhất hiện nay. Python được sử dụng để phát triển trang Web và phần mềm, tự động hóa tác vụ, phân tích dữ liệu và trực quan hóa dữ liệu.

Cùng với đó, hiện nay, đa số những người dùng mạng xã hội đều biết đến những con game kí thú với những trò chơi và lối chơi hấp dẫn, thú vị và kịch tính. Đến nay, tựa game FPS vẫn đứng hot trong các Bảng xếp hạng về game.

Với đề tài "**Lập trình Game bắn súng FPS bằng Python**" của chúng em thì chúng em sẽ hướng đến chi tiết nhất về trải nghiệm của người chơi con game này của chúng em. Với các thao tác đơn giản, con game này sẽ là đề tài mà chúng em sẽ phát triển lên.

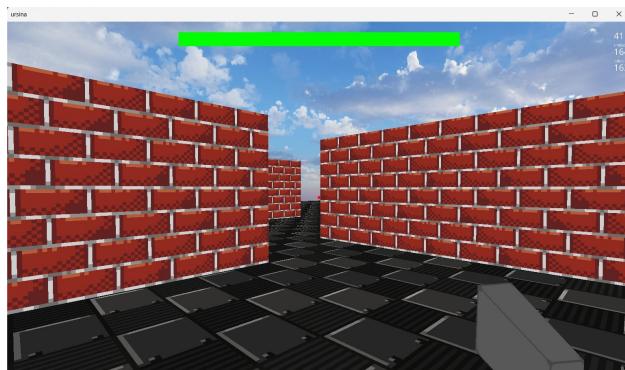
STT	Họ & Tên	Công Việc	Khối lượng làm việc
001	Nguyễn Minh Tú	Viết báo cáo và Phần Socket	33%
002	Nhữ Văn Tuyền	Viết báo cáo và Phần Game, Server	33%
003	Nguyễn Đăng Vũ	Viết báo cáo và Phần Game, Client	33%



3 Phần giới thiệu

3.1 Lý do chọn đề tài

Game bắn súng FPS (First-Person Shooter) là một tựa game hot - hầu như được đông đảo người chơi do tính hấp dẫn cũng như sự gây cấn, thú vị và lôi cuốn của nó. Với thiết kế của tựa game nhiều người chơi, trò chơi này sẽ mang lại sự kết nối của các người chơi khác nhau lại.



Hình 1: Giao diện Game

3.2 Giới thiệu về URSINA

Thư viện URSINA là một thư viện Python tuy mới nhưng rất mạnh mẽ, được thiết kế dùng để tạo ra các trò chơi 3D đơn giản và hiệu quả. Với một tốc độ biên dịch có thể nói là nhanh nhất bây giờ, URSINA có thể biên dịch chương trình tốt nhất. Dưới đây là hình ảnh minh họa về tốc độ chạy của URSINA.

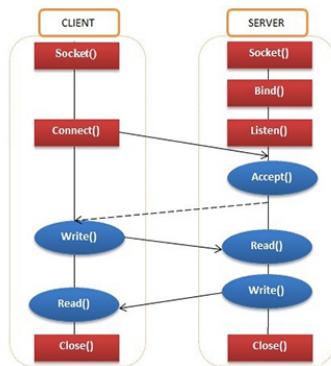
engine:	duration:	times as slow:
ursina	00.01s	1x
Godot	01.18s	118x
Unity	11.45s	1145x

Hình 2: Iterate Faster

3.3 Giới thiệu về Socket

Socket là giao diện lập trình ứng dụng mạng được dùng để truyền và nhận dữ liệu trên Internet. Giữa hai chương trình chạy trên mạng cần có một liên kết giao tiếp hai chiều (Two-way Communication) để kết nối 2 Process trò chuyện với nhau. Điểm cuối (Endpoint) của liên kết này được gọi là Socket.

Ngoài ra, Socket còn giúp các tầng TCP hoặc TCP Layer định danh ứng dụng mà dữ liệu sẽ được gửi tới thông qua sự ràng buộc với một cổng Port, từ đó tiến hành kết nối giữa Client và Server.



Hình 3: Mô hình Socket

3.4 Giới thiệu về chương trình

Game "Bắn súng FPS" của chúng em là một tựa game mô phỏng lại game bắn súng thông thường. Game được chơi theo thể loại sinh tồn (người này tiêu diệt người kia). Cũng như bao game khác, game này sẽ vẫn giữ các nút thao tác di chuyển như các game phổ thông. Với chế độ nhiều người chơi, chúng ta có thể ghép trận đấu với những người chơi khác và tiêu diệt họ.

Tựa game của chúng em tạo ra trải nghiệm thực tế đối với người chơi. Với các thao tác đơn giản để di chuyển và bắn súng, cho người dùng nhanh chóng tiếp cận và làm quen với cách chơi.

Với chất lượng hình ảnh cao và kết nối đa người dùng thì tựa game chúng em hướng đến sự giao lưu giữa người chơi với nhau.

4 Tổng quan chương trình

4.1 Game

4.1.1 Player

The screenshot shows a code editor with three tabs open:

- player.py**: A class `Player(first_person_controller)` is defined. It has methods for jumping, running, and shooting. It also handles collision detection with a health bar and a death message.
- first_person_controller.py**: A class `FirstPersonController` is defined. It has methods for setting up the camera, rendering, and updating the player's state. It also handles player controls like movement and shooting.
- setup.py**: A module that imports `setuptools` and `os`. It defines a `main()` function that sets up the game by calling `setup_main()`, `camera()`, and `setup_shaders()`. It then runs the game loop by calling `run_game()`. Finally, it runs the application with `if __name__ == "__main__": main()`.

Hình 4: Player

4.1.2 Setup

The screenshot shows the `setup.py` file from the previous image. It includes imports for `os` and `setuptools`. It defines a `main()` function that sets up the game by calling `camera()`, `setup_shaders()`, and `run_game()`. The `main()` function is the entry point of the application, indicated by the line `if __name__ == "__main__": main()` at the bottom.

Hình 5: Setup



4.1.3 Enemy

```
enemy.py * 
game = None
import urina

class Enemy(urina.Entity):
    def __init__(self, position: urina.Vector, identifier: str, username: str):
        super().__init__(position, identifier)
        self.username = username
        self.position = position
        self.y = position.y
        self.x = position.x
        self.z = 0
        self.origin_x = 0.5
        self.origin_y = 0.5
        self.origin_z = 0
        self.size_x = 0.5
        self.size_y = 0.5
        self.size_z = 0.5
        self.texture = "white_cube"
        self.urina_color_color((0, 0, 0, 1))
        self.urina_color_color((0, 0, 0, 1))

    def get_urina_entity(self):
        return urina.Entity(
            position=urina.Vector(0.5, 0.5, 0.5),
            size=urina.Vector(0.5, 0.5, 0.5),
            texture="white_cube",
            color=urina.color_color((0, 0, 0, 1))
        )

    def name_tag(self):
        return urina.Text(
            position=urina.Vector(0.5, 1.5, 0),
            text=f'{self.username} {self.identifier}',
            font_size=10,
            bold=True,
            italic=False,
            color=urina.color_color((0, 0, 0, 1))
        )

    def health(self):
        self.health -= 100
        self.id = identifier
        self.username = username
        self.name_tag()
        self.health -= 100
        self.id = identifier
        self.username = username
        self.name_tag()

    def update(self, set):
        self.health -= 100
        self.id = identifier
        self.username = username
        self.name_tag()
        self.health -= 100
        self.id = identifier
        self.username = username
        self.name_tag()

enemy.py (1) * 
game = None
import urina

class Enemy(urina.Entity):
    def __init__(self, position):
        super().__init__(position)
        self.y = position.y
        self.x = position.x
        self.z = 0
        self.color_saturate = lambda set: self.health / 100
        try:
            self.health = 100
            self.color_saturate = lambda set: self.health / 100
        except AttributeError:
            self.health = 100
            self.color_saturate = lambda set: self.health / 100
        self.texture = "white_cube"
        self.urina_color_color((0, 0, 0, 1))
        self.size_x = 0.5
        self.size_y = 0.5
        self.size_z = 0.5
        self.texture.filtering = True

    def color_saturate(self, set):
        if self.health < 0:
            self.health = 0
        self.urina_color_color(set)
```

Hình 6: Enemy

4.1.4 Floor

```
floor.py * 
game = None
import os
import urina

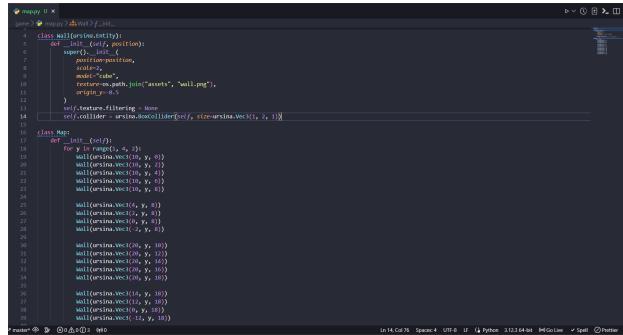
class Floor(urina.Entity):
    def __init__(self, position):
        super().__init__(position)
        self.y = position.y
        self.x = position.x
        self.z = 0
        self.size_x = 1
        self.size_y = 1
        self.size_z = 1
        self.texture = "white_cube"
        self.urina_color_color((0, 0, 0, 1))
        self.texture.filtering = True

    def check_darkness(self):
        dark1 = True
        for z in range(-1, 1, 2):
            for x in range(-1, 1, 2):
                for y in range(-1, 1, 2):
                    if self.get_urina_entity(urina.Vector(x, y, z)):
                        if dark1:
                            self.urina_color_color((0, 0, 0, 1))
                        else:
                            self.urina_color_color((0, 0, 0, 1))
                        dark1 = not dark1
                    else:
                        dark1 = not dark1
```

Hình 7: Floor



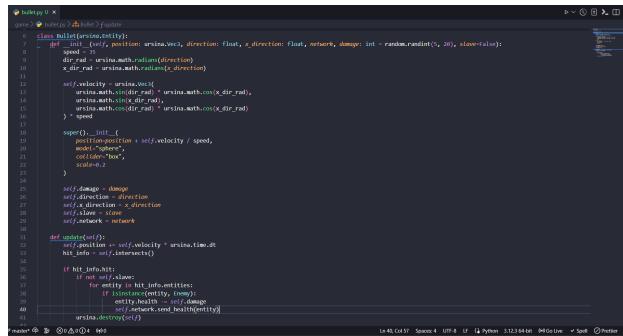
4.1.5 Map



```
map.py 16 lines, 1388 characters, 1388 bytes
1 import pygame
2
3 class Wall(pygame.sprite.Sprite):
4     def __init__(self, x, y):
5         super().__init__()
6         self.image = pygame.Surface([10, 10])
7         self.image.fill((0, 0, 0))
8         self.rect = self.image.get_rect()
9         self.rect.x = x
10        self.rect.y = y
11        self.mask = pygame.mask.from_surface(self.image)
12
13    def update(self, *args, **kwargs):
14        if self.rect.colliderect(*args[0]):
15            self.collider = pygame.sprite.collide_rect(*args[0], self))
16
17 class Map:
18     def __init__(self):
19         for y in range(1, 6, 2):
20             wallcoring.vec((0, y, 0))
21             wallcoring.vec((1, y, 0))
22             wallcoring.vec((2, y, 0))
23             wallcoring.vec((3, y, 0))
24             wallcoring.vec((4, y, 0))
25             wallcoring.vec((5, y, 0))
26             wallcoring.vec((6, y, 0))
27             wallcoring.vec((7, y, 0))
28             wallcoring.vec((8, y, 0))
29             wallcoring.vec((9, y, 0))
30             wallcoring.vec((10, y, 0))
31             wallcoring.vec((11, y, 0))
32             wallcoring.vec((12, y, 0))
33             wallcoring.vec((13, y, 0))
34             wallcoring.vec((14, y, 0))
35             wallcoring.vec((15, y, 0))
36             wallcoring.vec((16, y, 0))
37             wallcoring.vec((17, y, 0))
38             wallcoring.vec((18, y, 0))
39             wallcoring.vec((19, y, 0))
40             wallcoring.vec((20, y, 0))
41             wallcoring.vec((21, y, 0))
42             wallcoring.vec((22, y, 0))
43             wallcoring.vec((23, y, 0))
44             wallcoring.vec((24, y, 0))
45             wallcoring.vec((25, y, 0))
46             wallcoring.vec((26, y, 0))
47             wallcoring.vec((27, y, 0))
48             wallcoring.vec((28, y, 0))
49             wallcoring.vec((29, y, 0))
50             wallcoring.vec((30, y, 0))
51             wallcoring.vec((31, y, 0))
52             wallcoring.vec((32, y, 0))
53             wallcoring.vec((33, y, 0))
54             wallcoring.vec((34, y, 0))
55             wallcoring.vec((35, y, 0))
56             wallcoring.vec((36, y, 0))
57             wallcoring.vec((37, y, 0))
58             wallcoring.vec((38, y, 0))
59             wallcoring.vec((39, y, 0))
56
```

Hình 8: Map

4.1.6 Bullet



```
bullet.py 41 lines, 1639 characters, 1639 bytes
1 import pygame
2
3 class Bullet(pygame.sprite.Sprite):
4     def __init__(self, pos, direction, network, damage):
5         super().__init__()
6         self.image = pygame.Surface([5, 5])
7         self.image.set_colorkey((0, 0, 0))
8         self.image.fill((0, 0, 0))
9         self.rect = self.image.get_rect()
10        self.rect.x = pos[0]
11        self.rect.y = pos[1]
12        self.speed = 20
13        self.direction = direction
14        self.x_dir_rad = urina.math.radians(direction)
15        self.y_dir_rad = urina.math.radians(90 - direction)
16
17        self.velocity = urina.math.vec3(
18            urina.math.cos(self.x_dir_rad) * urina.math.cos(self.y_dir_rad),
19            urina.math.sin(self.x_dir_rad) * urina.math.cos(self.y_dir_rad),
20            urina.math.sin(self.y_dir_rad) * urina.math.cos(self.y_dir_rad)
21        )
22
23        self.damage = damage
24        self.position = pos
25        self.network = network
26        self.size = 2
27
28    def update(self):
29        self.rect.x += self.velocity[0] * urina.time.dt
30        self.rect.y += self.velocity[1] * urina.time.dt
31
32        hit_info = self.rect.intersect(network)
33
34        if hit_info:
35            for hit_info in hit_info:
36                if hit_info.entity == self:
37                    if hit_info.entity.is_temp:
38                        hit_info.entity.is_temp = False
39                        hit_info.entity.health -= self.damage
40                        if hit_info.entity.health <= 0:
41                            urina.destroy(hit_info.entity)
42
43
```

Hình 9: Bullet



4.1.7 Network

```
network.py
game > network >
1 import socket
2 import json
3
4 from player import Player
5 from osse import Osse
6 from bullet import Bullet
7
8
9 class Network:
10     """
11     A client class to abstract away socket functions and make communication with server less of a headache.
12     """
13
14     def __init__(self, server_ip="127.0.0.1", port=5555):
15         self.ip = server_ip
16         self.port = port
17         self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
18
19         self.username = None
20
21     def connect(self, username):
22         self.username = username
23         self.client.connect((self.ip, self.port))
24         self.recv_size = 1024
25
26     def get_server_info(self):
27         try:
28             self.client.settimeout(1)
29             self.client.recv(self.recv_size)
30         except socket.timeout:
31             print("Connection timed out")
32
33         self.client.send(f"({self.ip}, {self.port})".encode("utf-8"))
34         self.client.send(self.username.encode("utf-8"))
35
36     def get_message(self):
37         try:
38             self.client.settimeout(1)
39             msg = self.client.recv(self.recv_size)
40             if msg:
41                 print(msg)
42             else:
43                 return None
44
45             msg_decoded = msg.decode("utf-8")
46
47             left_bracket_index = msg_decoded.index("{")
48             right_bracket_index = msg_decoded.index("}") + 1
49             msg_decoded = msg_decoded[left_bracket_index:right_bracket_index]
50
51             msg_json = json.loads(msg_decoded)
52
53             msg_json["msg"] = msg
54
55             return msg_json
56
57         except socket.error as e:
58             print(e)
59
60     def send_player(self, player: Player):
61         player_info = {
62             "id": player.id,
63             "position": (player.world_x, player.world_y, player.world_z),
64             "rotation": player.rotation,
65             "health": player.health,
66             "is_alive": player.is_alive(),
67             "left": player.left,
68             "right": player.right
69         }
70
71         player_info_encoded = json.dumps(player_info).encode("utf-8")
72
73         try:
74             self.client.send(player_info_encoded)
75         except socket.error as e:
76             print(e)
77
78
bullet.py
game > network >
1 import socket
2 import json
3
4 from bullet import Bullet
5
6
7 class Network:
8     """
8     A client class to abstract away socket functions and make communication with server less of a headache.
9     """
10
11     def __init__(self, bullet):
12         self.bullet = bullet
13
14     def send_bullet(self, bullet):
15         bullet_info = {
16             "object": "bullet",
17             "world_x": bullet.world_x,
18             "world_y": bullet.world_y,
19             "world_z": bullet.world_z,
20             "damage": bullet.damage,
21             "direction": bullet.direction,
22             "x": bullet.x,
23             "y": bullet.y,
24             "z": bullet.z
25         }
26
27         bullet_info_encoded = json.dumps(bullet_info).encode("utf-8")
28
29         try:
30             self.client.send(bullet_info_encoded)
31         except socket.error as e:
32             print(e)
33
34
35
player.py
game > network >
1 import socket
2 import json
3
4 from osse import Osse
5
6
7 class Network:
8     """
8     A client class to abstract away socket functions and make communication with server less of a headache.
9     """
10
11     def __init__(self):
12         self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
13
14     def send_health(self, osse: Osse):
15         health_info = {
16             "object": "osse",
17             "health": osse.health,
18             "id": osse.id
19         }
20
21         health_info_encoded = json.dumps(health_info).encode("utf-8")
22
23         try:
24             self.client.send(health_info_encoded)
25         except socket.error as e:
26             print(e)
```

Hình 10: Network



4.1.8 Main

```
server.py
game.py
enemy.py
player.py
```

The terminal window displays four Python files:

- server.py**:
Imports socket, sys, select, and threading modules. It defines a function `main()` that reads server address and port from command-line arguments. It then enters a loop where it receives connections from clients and handles them.
- game.py**:
Imports socket, struct, and threading modules. It defines a function `main()` that handles client connections. It checks for errors and handles connection timeouts. It also manages a list of clients and their information.
- enemy.py**:
Imports random, math, and struct modules. It defines a function `main()` that handles enemy logic. It updates enemy positions, directions, and health based on player input and enemy movement rules.
- player.py**:
Imports random, math, and struct modules. It defines a function `main()` that handles player logic. It updates player positions, directions, and health based on player input and enemy interaction.



```
game> main.py < />
113
114     def update():
115         if player.health > 0:
116             if player.world_pos.y == prev_world_pos.y:
117                 n.send_player(player)
118             new_pos = player.world.position
119             n.send_player(player)
120             prev_pos = player.world.rotation_y
121             prev_dir = player.world.rotation_y
122
123     def input(key):
124         if key == 'left mouse down' and player.health > 0:
125             b_pos = player.world.pivot + player.world.rotation_x, 0
126             bullet_id = n.create_bullet(bullet_id, player.world.rotation_x, player.camera.pivot.world_rotation_x, 0)
127             n.send_bullet(bullet_id, bullet_id, 0)
128             weapon.destroy_bullet(bullet_id)
129
130
131     def main():
132         msg_threading = threading.Thread(target=receive, daemon=True)
133         msg_threading.start()
134         app.run()
135
136
137     if __name__ == "__main__":
138         main()
```

Hình 11: main

4.2 Server

4.2.1 Main

```
main> main.py < />
1 import socket
2 import json
3 import time
4 import random
5 from threading import Thread
6
7 CODE = "192.168.204.1"
8 PORT = 8000
9 MAX_PLAYERS = 10
10 MSG_SIZE = 2048
11
12
13 # Setup server socket
14 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
15 s.bind((CODE, PORT))
16 s.listen(MAX_PLAYERS)
17
18 players = []
19
20 def generate_id(player_list: dict, max_players: int):
21     """Generate a unique identifier
22     """
23     args: tuple = player_list.items()
24     max_players = max_players - len(args)
25     return max_players
26
27     return
28
29 while True:
30     unique_id = str(random.randint(1, max_players))
31     unique_id not in player_list
32     unique_id
```



```
main> handle.py < />
1 def handle_message(client_info: str):
2     """Handle messages from clients
3     """
4     conn, client_socket = client_info["socket"]
5     username = client_info["username"]
6
7     while True:
8
9         msg = conn.recv(MSG_SIZE)
10        conn.setblocking(True)
11        break
12
13        if not msg:
14            break
15
16        msg_decoded = msg.decode("utf-8")
17
18        try:
19            left_bracket_index = msg_decoded.index("(") + 1
20            right_bracket_index = msg_decoded.index(")") + 1
21            msg_decoded = msg_decoded[left_bracket_index:right_bracket_index]
22            command = msg_decoded[0]
23            continue
24
25        try:
26            msg_decoded = json.loads(msg_decoded)
27        except Exception as e:
28            print(e)
29            continue
30
31        print(f"Received message from player {username} with ID {unique_id}")
32
33        if command == "position":
34            player[unique_id][position] = msg_decoded["position"]
35            player[unique_id][rotation] = msg_decoded["rotation"]
36            player[unique_id][health] = msg_decoded["health"]
```



The image shows three separate terminal windows side-by-side, each displaying the same Python code for a network server. The code is written in Python 3 and uses the socket module for networking.

```
server > cd /Users/.../Desktop/...
server > python main.py
[...]
def handle_message(data):
    if player_id in players:
        player_info = players[player_id]
        player_com = socket.socket(player_info['socket'])
        player_com.sendall(msg_decoded.encode("utf-8"))
        except OSError:
            pass
    else:
        print(f"Player {username} with ID {player_id} has left the game...")
        del players[player_id]
        del player_info[player_id]
        conn.close()
    if len(players) == 0:
        print("Server started, listening for new connections...")
        while True:
            accept_new_connection_and_assign_unique_id()
            new_id = len(players)
            if new_id >= MAX_PLAYERS:
                conn.sendall(str.encode("Full"))
                break
            new_player_info = {
                "socket": conn,
                "username": username,
                "position": (0, 1, 0),
                "rotation": 0,
                "health": 100
            }
            new_player.info = new_player_info
            new_player.info['socket'].sendall(str.encode("Full"))
            new_player.info['socket'].recv(1024).decode('utf-8')
            new_player.info['socket'].close()
            conn.close()
            print(f"New player about to exist players")
            for player_id in players:
                if player_id != new_id:
                    player_info = players[player_id]
                    try:
                        conn.sendall(str.encode("Full"))
                        msg = conn.recv(1024).decode('utf-8')
                        if msg == "Full":
                            player_info['socket'].close()
                            conn.close()
                            break
                        else:
                            print("Received message from player " + str(player_id))
                            print("Message: " + msg)
                            if msg == "QUIT":
                                player_info['socket'].close()
                                conn.close()
                                break
                            else:
                                print("Message received from player " + str(player_id))
                                print("Message: " + msg)
                    except OSError:
                        pass
            if len(players) == 0:
                print("No more players effectively allowing it to receive messages from other players")
                break
            else:
                new_player_id = new_id
                player_info = players[new_player_id]
                conn.sendall(str.encode("Full"))
                msg = conn.recv(1024).decode('utf-8')
                if msg == "Full":
                    player_info['socket'].close()
                    conn.close()
                    break
                else:
                    print("Received message from player " + str(new_player_id))
                    print("Message: " + msg)
                    if msg == "QUIT":
                        player_info['socket'].close()
                        conn.close()
                        break
                    else:
                        print("Message received from player " + str(new_player_id))
                        print("Message: " + msg)
    print("New connection from ", addr, " assigned ID: ", new_id,...)

if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        pass
    except OSError:
        pass
    finally:
        print("Exiting")
    if __name__ == "__main__":
        exit(0)
```

Hình 12: Main



5 Kết luận

Trò chơi FPS bắn súng không chỉ là một thách thức giải trí mà còn là một cơ hội tuyệt vời để thử nghiệm và phát triển kỹ năng lập trình Python. Việc tạo ra các tính năng như hệ thống điều khiển nhân vật, cơ chế va chạm và hiệu ứng đồ họa đều đòi hỏi kiến thức sâu rộng về Python và các thư viện như Pygame.

Mỗi dòng code được viết trong trò chơi này không chỉ là một bước tiến trong quá trình phát triển, mà còn là một cơ hội để thử nghiệm, sáng tạo và học hỏi từ các lập trình viên khác. Thông qua việc tương tác với cộng đồng lập trình Python, người chơi có thể chia sẻ ý tưởng, hỏi đáp về vấn đề kỹ thuật và hợp tác trong việc xây dựng trò chơi phức tạp hơn.

Tóm lại, trò chơi FPS bắn súng không chỉ là một trải nghiệm giải trí mà còn là một cơ hội để khám phá và ứng dụng kiến thức lập trình Python vào thực tế, đồng thời tạo ra một môi trường sáng tạo và học hỏi trong cộng đồng lập trình viên.



6 Tài liệu tham khảo

Tài liệu

- [1] Tài liệu về Ursina.
- [2] Tài liệu về Socket.
- [3] Tài liệu về Pygame.