

TÁC GIẢ: 8 SYNC DEV

## GIẢI THÍCH CHI TIẾT TỪNG DÒNG CODE VÀ CẤU TRÚC DỰ ÁN

---

### CẤU TRÚC DỰ ÁN

{ 8 Sync Dev }

```
my_project/  
├─ main.py  
├─ src/  
│   ├─ entity/  
│   │   ├─ character.py  
│   │   ├─ scene.py  
│   │   └─ event.py  
│   └─ network/  
│       ├─ client.py  
│       └─ server.py  
├─ resources.py  
├─ settings.py  
└─ requirements.txt
```

### MÔ TẢ CẤU TRÚC

- **main.py**: Tập chính để chạy ứng dụng.
- **src/**: Thư mục chứa mã nguồn chính của dự án.

- **entity/**: Thư mục chứa các module liên quan đến các thực thể trong trò chơi.
  - **character.py**: Định nghĩa các lớp liên quan đến nhân vật (Player).
  - **scene.py**: Định nghĩa các lớp liên quan đến cảnh và khối (Block).
  - **event.py**: Định nghĩa các sự kiện trò chơi.
- **network/**: Thư mục chứa các module liên quan đến mạng.
  - **client.py**: Mã nguồn liên quan đến client.
  - **server.py**: Mã nguồn liên quan đến server.
- **resources.py**: Quản lý các tài nguyên trò chơi.
- **settings.py**: Cấu hình các thiết lập cho dự án.
- **requirements.txt**: Danh sách các thư viện phụ thuộc cần thiết cho dự án.

## MAIN.PY

{ 8 Sync Dev }

```
from src.settings import BASE_DIR
from ursina import *

app = Ursina()
from src.resources import * # tải tất cả các tài nguyên
from src.network.client import ClientService
ClientService.create_services()

from src.entity.scene import hand
# from src.entity.genmap import gen_map
# gen_map()
# from src.entity.character import character
from src.entity.event import update, input
mouse.locked = False
window.borderless = False
app.run()
```

- **Mục đích:** Tập chính để chạy ứng dụng, thiết lập môi trường trò chơi và khởi chạy client.
- **Chi tiết:**
  - `from src.settings import BASE_DIR`: Nhập biến `BASE_DIR` từ tệp `settings.py`.
  - `from ursina import *`: Nhập tất cả các chức năng từ thư viện `ursina`.
  - `app = Ursina()`: Khởi tạo ứng dụng `Ursina`.
  - `from src.resources import *`: Nhập tất cả các tài nguyên từ tệp `resources.py`.
  - `from src.network.client import ClientService`: Nhập lớp `ClientService` từ tệp `client.py`.
  - `ClientService.create_services()`: Tạo các dịch vụ client.
  - `from src.entity.scene import hand`: Nhập `hand` từ tệp `scene.py`.
  - `mouse.locked = False`: Không khóa chuột trong trò chơi.
  - `window.borderless = False`: Hiển thị cửa sổ trò chơi với viền.
  - `app.run()`: Chạy ứng dụng.

## CLIENT.PY



{ 8 Sync Dev }

```
from ursinanetworking import (  
    UrsinaNetworkingClient,  
    EasyUrsinaNetworkingClient,  
)
```

- **Mục đích:** Nhập các lớp `UrsinaNetworkingClient` và `EasyUrsinaNetworkingClient` từ thư viện `ursinanetworking`.
- **Chức năng:** Cung cấp các phương tiện để kết nối và giao tiếp với máy chủ trong ứng dụng `Ursina`.



{ 8 Sync Dev }



```
from src.entity.character import Player, PlayerPresentation
```

- **Mục đích:** Nhập các lớp `Player` và `PlayerPresentation` từ module `character`.
- **Chức năng:** Quản lý thông tin và hiển thị của các đối tượng `Player` trong trò chơi.



{ 8 Sync Dev }

```
from ursina import color, destroy
```

- **Mục đích:** Nhập các hàm `color` và `destroy` từ thư viện `ursina`.
- **Chức năng:** `color` để thiết lập màu sắc cho các đối tượng và `destroy` để xóa các đối tượng khỏi cảnh.



{ 8 Sync Dev }

```
from ursina import Vec3
```

- **Mục đích:** Nhập lớp `Vec3` từ thư viện `ursina`.
- **Chức năng:** Quản lý các vector 3D để xử lý vị trí và chuyển động trong không gian 3D.



{ 8 Sync Dev }

```
from src.entity.scene import Block
import random
```

```
ClientId = -1
PlayersTargetPosition = {}
Players = {}
```

- **Mục đích:** Nhập lớp `Block` từ module `scene` và sử dụng module `random`.
- **Chức năng:** Quản lý các khối và các đối tượng `Player` trong trò chơi.
- `ClientId = -1`: Biến toàn cục để lưu trữ ID của client.
- `PlayersTargetPosition = {}`: Từ điển để lưu trữ vị trí mục tiêu của các player.
- `Players = {}`: Từ điển để lưu trữ các đối tượng `Player`.

{ 8 Sync Dev }

```
class BlockState:  
    count = 0  
    blocks = {}
```

- **Mục đích:** Định nghĩa lớp `BlockState` để quản lý trạng thái của các khối.
- **Chi tiết:**
  - `count = 0`: Đếm số lượng khối.
  - `blocks = {}`: Từ điển để lưu trữ các khối.

{ 8 Sync Dev }

```
user = Player()
```

- **Mục đích:** Khởi tạo một đối tượng `Player`.
- **Chức năng:** Quản lý thông tin và hành vi của player.

{ 8 Sync Dev }

```
client = UrsinaNetworkingClient('localhost', 25565)  
easy_client = EasyUrsinaNetworkingClient(client)
```

- **Mục đích:** Khởi tạo các đối tượng client để kết nối với máy chủ.

- **Chi tiết:**

- `client`: Kết nối với máy chủ tại địa chỉ `localhost` và cổng `25565`.
- `easy_client`: Cung cấp một lớp đơn giản hơn để quản lý client.



{ 8 Sync Dev }

```
Block.client = client
```

- **Mục đích:** Gán đối tượng `client` cho thuộc tính `client` của lớp `Block`.
- **Chức năng:** Để các khối có thể giao tiếp với máy chủ.



{ 8 Sync Dev }

```
class ClientService:  
    @staticmethod  
    def create_services():
```

- **Mục đích:** Định nghĩa lớp `ClientService` để quản lý các dịch vụ của client.
- **Chức năng:** Cung cấp các phương thức tĩnh để tạo và chạy các dịch vụ client.



{ 8 Sync Dev }

```
@client.event  
def onConnectionEstablished():  
    print('Connected to server')
```

- **Mục đích:** Đăng ký một sự kiện khi kết nối đến máy chủ được thiết lập.
- **Chức năng:** In ra thông báo "Connected to server" khi kết nối thành công.



{ 8 Sync Dev }



```
@client.event
def onConnectionError(reason):
    print(f'Connection error: {reason}')
```

- **Mục đích:** Đăng ký một sự kiện khi xảy ra lỗi kết nối.
- **Chức năng:** In ra thông báo lỗi kết nối cùng với lý do.



{ 8 Sync Dev }

```
@client.event
def connected(data):
    global ClientId
    print('Data', data)
    ClientId = data['id']
```

- **Mục đích:** Đăng ký một sự kiện khi nhận được thông tin kết nối từ máy chủ.
- **Chức năng:** In ra dữ liệu nhận được và lưu trữ ID của client.



{ 8 Sync Dev }

```
@easy_client.event
def onReplicatedVariableCreated(res_server):
    var_name = res_server.name
    type_name = res_server.content['type']
```

- **Mục đích:** Đăng ký một sự kiện khi một biến được tạo trên máy chủ.
- **Chức năng:** Xử lý các biến được tạo và thêm chúng vào từ điển tương ứng.



{ 8 Sync Dev }

```
if type_name == 'player':
    PlayersTargetPosition[var_name] = Vec3(0, 0, 0)
```

```

        Players[var_name] = PlayerPresentation()
        if ClientId ==
int(res_server.content['client_id']):
            Players[var_name].color = color.yellow
            Players[var_name].visible = False

```

- **Mục đích:** Xử lý khi loại biến là **player**.
- **Chức năng:** Tạo một đối tượng **PlayerPresentation**, thiết lập màu sắc và trạng thái hiển thị cho player hiện tại.



{ 8 Sync Dev }

```

        elif type_name == 'block':
            BlockState.blocks[var_name] =
Block(res_server.content['position'], block_name=var_name)
            BlockState.count += 1

```

- **Mục đích:** Xử lý khi loại biến là **block**.
- **Chức năng:** Tạo một đối tượng **Block**, thêm vào từ điển **blocks** và tăng số lượng khối.



{ 8 Sync Dev }

```

@easy_client.event
def onReplicatedVariableUpdated(res_server):
    PlayerPresentation[res_server.name] =
res_server.content['position']

```

- **\*\*Mục đích**
- \*\*:** Đăng ký một sự kiện khi một biến được cập nhật trên máy chủ.
- **Chức năng:** Cập nhật vị trí của player trong từ điển **PlayerPresentation**.



{ 8 Sync Dev }



```
@easy_client.event
def onReplicatedVariableRemoved(res_server):
    var_name = res_server.name
    type_name = res_server.content['type']
```

- **Mục đích:** Đăng ký một sự kiện khi một biến bị xóa trên máy chủ.
- **Chức năng:** Xử lý việc xóa các biến khỏi từ điển tương ứng.

{ 8 Sync Dev }

```
if type_name == 'player':
    PlayersTargetPosition.pop(var_name)
    Players.pop(var_name)
```

- **Mục đích:** Xử lý khi loại biến là **player**.
- **Chức năng:** Xóa player khỏi các từ điển **PlayersTargetPosition** và **Players**.

{ 8 Sync Dev }

```
elif type_name == 'block':
    destroy(BlockState.blocks[var_name])
    BlockState.blocks.pop(var_name)
    BlockState.count -= 1
```

- **Mục đích:** Xử lý khi loại biến là **block**.
- **Chức năng:** Xóa khối khỏi cảnh và từ điển **blocks**, giảm số lượng khối.

{ 8 Sync Dev }

```
@staticmethod
def run_client():
```

```
easy_client.process_net_events()
```

- **Mục đích:** Định nghĩa phương thức để xử lý các sự kiện mạng của client.
- **Chức năng:** Gọi phương thức `process_net_events` của `easy_client`.

{ 8 Sync Dev }

```
def run_client():  
    if user.position[1] < -5:  
        user.position = (random.randrange(0, 20), 20,  
random.randrange(0, 20))
```

- **Mục đích:** Định nghĩa hàm để chạy client.
- **Chức năng:** Kiểm tra và đặt lại vị trí của player nếu vị trí y nhỏ hơn -5.

{ 8 Sync Dev }

```
for player_id, target_position in  
PlayersTargetPosition.items():  
    Players[player_id].position += (Vec3(target_position) -  
Players[player_id].position) / 20
```

- **Mục đích:** Cập nhật vị trí của tất cả các player.
- **Chức năng:** Tính toán và di chuyển player đến vị trí mục tiêu.

{ 8 Sync Dev }

```
ClientService.run_client()
```

- **Mục đích:** Gọi phương thức `run_client` của `ClientService`.
- **Chức năng:** Xử lý các sự kiện mạng của client.

## { 8 Sync Dev }

```
def input_client(key):  
    client.send_message('get_client_position', user.position +  
        (0, 1, 0))
```

- **Mục đích:** Định nghĩa hàm để gửi vị trí của client khi nhận input.
- **Chức năng:** Gửi thông điệp `get_client_position` với vị trí của player đến máy chủ.

## SERVER.PY

## { 8 Sync Dev }

```
from ursinaNetworking import (  
    UrsinaNetworkingServer,  
    UrsinaNetworkingConnectedClient,  
    EasyUrsinaNetworkingServer,  
)  
  
from src.utils.noise import gen_noise
```

- **Mục đích:** Nhập các lớp và hàm cần thiết từ thư viện `ursinaNetworking` và module `utils.noise`.
- **Chức năng:** Cung cấp các phương tiện để quản lý máy chủ và tạo noise cho bản đồ.

## { 8 Sync Dev }

```
server = UrsinaNetworkingServer('localhost', 25565)  
easy_server = EasyUrsinaNetworkingServer(server)
```



- **Mục đích:** Khởi tạo các đối tượng server để lắng nghe kết nối từ client.
- **Chức năng:**
  - `server`: Lắng nghe tại địa chỉ `localhost` và cổng `25565`.
  - `easy_server`: Cung cấp một lớp đơn giản hơn để quản lý server.

{ 8 Sync Dev }

```
class BlockState:  
    count = 0  
    blocks = {}
```

- **Mục đích:** Định nghĩa lớp `BlockState` để quản lý trạng thái của các khối.
- **Chi tiết:**
  - `count = 0`: Đếm số lượng khối.
  - `blocks = {}`: Từ điển để lưu trữ các khối.

{ 8 Sync Dev }

```
class ServerService:  
    @staticmethod  
    def create_services():
```


- **Mục đích:** Định nghĩa lớp `ServerService` để quản lý các dịch vụ của server.
- **Chức năng:** Cung cấp các phương thức tĩnh để tạo và chạy các dịch vụ server.

{ 8 Sync Dev }

```
@server.event  
def onClientConnected(client:  
    UrsinaNetworkingConnectedClient):  
    print(f'Client {client.id} connected')
```

```
easy_server.create_replicated_variable(f'player_{client.id}', {
    'type': 'player',
    'position': (0, 0, 0),
    'client_id': client.id
})
client.send_message('connected', {'id': client.id})
```

- **Mục đích:** Đăng ký một sự kiện khi client kết nối.
- **Chức năng:** Tạo biến nhân bản `player` cho client và gửi thông điệp kết nối với ID của client.




{ 8 Sync Dev }

```
@server.event
def onClientDisconnected(client:
UrsinaNetworkingConnectedClient):
    print(f'Client {client.id} disconnected')

easy_server.remove_replicated_variable_by_name(f'player_{client.
id}')
```

- **Mục đích:** Đăng ký một sự kiện khi client ngắt kết nối.
- **Chức năng:** Xóa biến nhân bản `player` của client.



{ 8 Sync Dev }

```
@server.event
def request_destroy_block(client:
UrsinaNetworkingConnectedClient, name_block: str):
    ServerService.remove_block(name_block)
```

- **Mục đích:** Đăng ký một sự kiện khi client yêu cầu xóa khối.
- **Chức năng:** Gọi phương thức `remove_block` để xóa khối theo tên.

### { 8 Sync Dev }

```
@server.event
def request_create_block(client:
UrsinaNetworkingConnectedClient, position: tuple):
    ServerService.create_block(position)
```

- **Mục đích:** Đăng ký một sự kiện khi client yêu cầu tạo khối.
- **Chức năng:** Gọi phương thức `create_block` để tạo khối tại vị trí yêu cầu.

### { 8 Sync Dev }

```
@server.event
def get_client_position(client:
UrsinaNetworkingConnectedClient, position: tuple):

easy_server.update_replicated_variable_by_name(f'player_{client.
id}', 'position', position)
```

- **Mục đích:** Đăng ký một sự kiện khi nhận được vị trí của client.
- **Chức năng:** Cập nhật vị trí của `player` trên server.

### { 8 Sync Dev }

```
ServerService.gen_map()

ServerService.run_server()
```

- **Mục đích:** Tạo bản đồ và chạy server.
- **Chức năng:** Gọi phương thức `gen_map` để tạo bản đồ và `run_server` để chạy server.



## { 8 Sync Dev }

```
@staticmethod
def run_server():
    while True:
        easy_server.process_net_events()
```

- **Mục đích:** Định nghĩa phương thức để chạy server.
- **Chức năng:** Xử lý các sự kiện mạng của server trong một vòng lặp vô hạn.

## { 8 Sync Dev }

```
@staticmethod
def create_block(position):
    count = BlockState.count
    block_name = f'block_{count}'
    easy_server.create_replicated_variable(block_name, {
        'type': 'block',
        'position': position,
        'name': block_name
    })

    BlockState.blocks[block_name] = {
        'type': 'block',
        'position': position,
        'name': block_name
    }
    BlockState.count += 1
```

- **Mục đích:** Định nghĩa phương thức để tạo khối.
- **Chức năng:** Tạo biến nhân bản **block** và thêm khối vào từ điển **blocks**.

## { 8 Sync Dev }

```
@staticmethod
def remove_block(name):
    easy_server.remove_replicated_variable_by_name(name)
    BlockState.blocks.pop(name)
```

- **Mục đích:** Định nghĩa phương thức để xóa khối.
- **Chức năng:** Xóa biến nhân bản `block` và loại bỏ khối khỏi từ điển `blocks`.

```
@staticmethod
def gen_map():
    for z in range(30):
        for x in range(30):
            y = int(gen_noise()([x * 0.1, z * 0.1]) * 10)
            ServerService.create_block((x, y, z))
```

- **Mục đích:** Định nghĩa phương thức để tạo bản đồ.
- **Chức năng:** Tạo khối tại các vị trí được tạo bởi hàm `noise`.

```
ServerService.create_services()
```

- **Mục đích:** Gọi phương thức để tạo các dịch vụ server.
- **Chức năng:** Khởi tạo các sự kiện và chạy server.

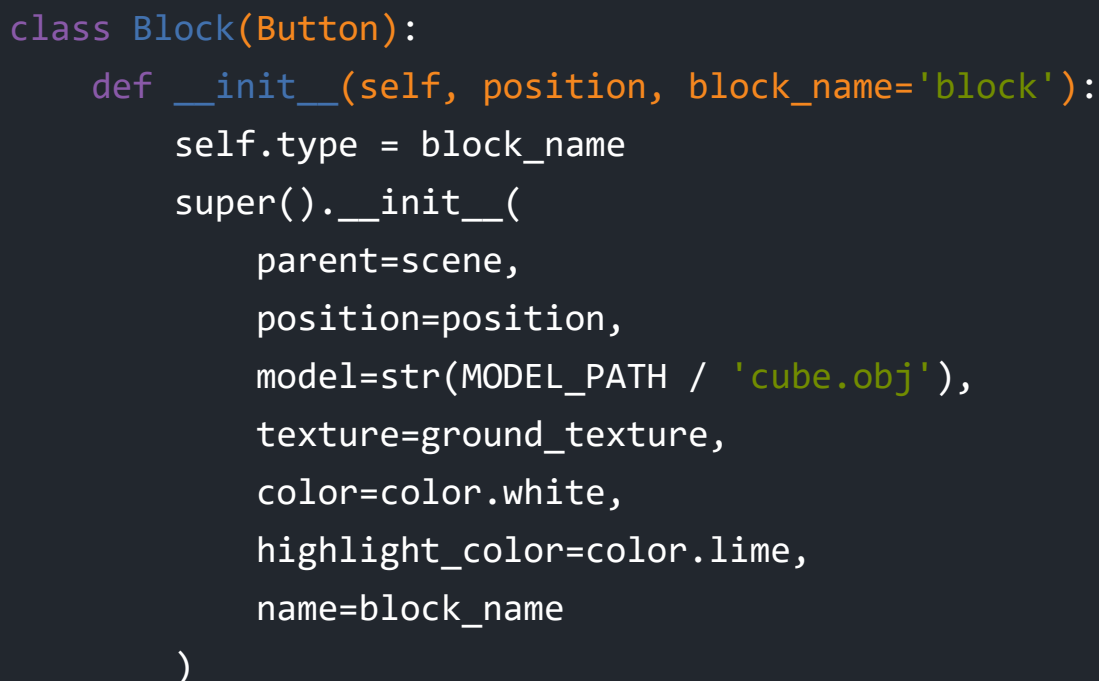
BLOCK.PY

```
from ursina import (  
    Button,  
    destroy,  
    scene,  
    color,  
    mouse,  
)  
import random  
from src.resources import *  
from src.settings import MODEL_PATH
```

- **Mục đích:** Nhập các lớp

và hàm từ thư viện **ursina** và các module trong dự án.

- **Chức năng:** Quản lý các khối trong trò chơi.



```
class Block(Button):  
    def __init__(self, position, block_name='block'):  
        self.type = block_name  
        super().__init__(  
            parent=scene,  
            position=position,  
            model=str(MODEL_PATH / 'cube.obj'),  
            texture=ground_texture,  
            color=color.white,  
            highlight_color=color.lime,  
            name=block_name  
        )
```

- **Mục đích:** Định nghĩa lớp **Block** để quản lý các khối.
- **Chức năng:** Thiết lập thuộc tính và phương thức cho khối.
- **Chi tiết:**
  - **self.type = block\_name:** Gán tên cho khối.



- `super().__init__(...)`: Gọi phương thức khởi tạo của lớp `Button` để thiết lập khối.

{ 8 Sync Dev }

```
@staticmethod
def random_color():
    return color.rgb(
        random.randrange(256),
        random.randrange(256),
        random.randrange(256),
    )
```

- **Mục đích:** Định nghĩa phương thức để tạo màu ngẫu nhiên.
- **Chức năng:** Trả về một màu ngẫu nhiên sử dụng hàm `rgb` của `color`.

{ 8 Sync Dev }

```
@staticmethod
def handle_block(position):
    for block in scene.entities:
        if block.position == position:
            destroy(block)
    return
    Block(position=position)
```

- **Mục đích:** Định nghĩa phương thức để xử lý khối tại vị trí nhất định.
- **Chức năng:** Nếu khối tồn tại, xóa khối, ngược lại, tạo khối mới tại vị trí đó.

## CỘNG ĐỒNG

## KHÓA HỌC:



[Kevin Nguyễn](#)



[Fullstack Python](#)



[Nhóm Chia Sẻ Công Nghệ](#)



[Fullstack Nextjs](#)



[Nhóm BlockChain](#)



[Fullstack Android-IOS](#)



[Tiktok: 8 Sync](#)



[Youtube: 8 Sync Dev](#)



[Zalo](#)

Tài liệu sẽ được cập nhật định kì và thông báo trong group nên các bạn chú ý nhen .

Các khóa học khác xem tại : [8 Sync Dev](#)

Hoặc liên hệ trực tiếp qua fb: [Kevin Nguyễn](#) để được hỗ trợ

