

TRƯỜNG ĐẠI HỌC KỸ THUẬT CÔNG NGHIỆP

KHOA ĐIỆN TỬ

Bộ Môn: Công Nghệ Thông Tin



**BÀI TẬP LỚN**

MÔN HỌC

**LẬP TRÌNH GAME 3D VỚI UNITY**

*Đề tài: Game shooter 3D unity multiplayer*

Sinh viên:

NGUYỄN THỌ DUY

NGUYỄN VĂN HƯNG

LÊ THỊ THU TRANG

HOÀNG THANH TÙNG

Lớp:

55KMT

GVHD:

Th.S ĐỖ DUY CỚP

**Thái Nguyên – 2023**

**TRƯỜNG ĐHKTCN CỘNG HOÀ XÃ HỘI CHỦ NGHĨA VIỆT NAM**

**KHOA ĐIỆN TỬ**

**Độc lập - Tự do - Hạnh phúc**

## **BÀI TẬP LỚN**

**MÔN HỌC: LẬP TRÌNH GAME 3D VỚI UNITY**

**BỘ MÔN : CÔNG NGHỆ THÔNG TIN**

*Sinh viên : Nguyễn Thọ Duy, Nguyễn Văn Hưng,*

*Lê Thị Thu Trang, Hoàng Thanh Tùng*

*Lớp : 55KMT*

*Ngành : Kỹ thuật máy tính*

*Ngày giao đề : 15/05/2023*

*Ngày hoàn thành : 10/06/2023*

1. Tên đề tài : Game shooter 3D unity multiplayer

2. Nội dung:

- Giới thiệu về đề tài, Tổng quan về unity, game FPS Multiplayer
- Phân tích và thiết kế
- Xây dựng chương trình và kiểm thử
- Kết luận

3. Các bản vẽ, chương trình và đồ thị: Kiểm thử chương trình

**TRƯỞNG BỘ MÔN**

**GIÁO VIÊN HƯỚNG DẪN**

*(Ký và ghi rõ họ tên)*

*(Ký và ghi rõ họ tên)*

## NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

*Thái Nguyên, ngày....tháng.....năm 2023*

**GIÁO VIÊN HƯỚNG DẪN**

(Ký ghi rõ họ tên)

## NHẬN XÉT CỦA GIÁO VIÊN CHĂM

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

*Thái Nguyên, ngày... tháng.....năm 2023*

**GIÁO VIÊN CHĂM**

*(Ký ghi rõ họ tên)*

## PHÂN CÔNG CÔNG VIỆC

Nguyễn Thọ Duy	Liên kết các người chơi
Nguyễn Văn Hưng	Điều khiển súng
Lê Thị Thu Trang	Thiết kế bản đồ, kịch bản
Hoàng Thanh Tùng	Điều khiển nhân vật và di chuyển

## **sMỤC LỤC**

<b>PHÂN CÔNG CÔNG VIỆC .....</b>	<b>5</b>
<b>DANH MỤC HÌNH ẢNH.....</b>	<b>8</b>
<b>LỜI NÓI ĐẦU .....</b>	<b>9</b>
<b>CHƯƠNG 1: TỔNG QUAN.....</b>	<b>10</b>
1.1. Giới thiệu về đề tài.....	10
1.2. Tổng quan về Unity.....	10
1.2.1. Đối tượng tham gia hệ thống.....	10
1.2.2. Lịch sử của Unity .....	11
1.3. Tổng quan về Photonengine.....	12
1.4. Game FPS Multiplayer .....	15
<b>CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ .....</b>	<b>16</b>
2.1. Khảo sát hệ thống.....	16
2.2. Thông tin vào ra của hệ thống. ....	17
2.3. Phân tích game.....	18
2.3.1. Cách chơi.....	18
2.3.2. Bối cảnh game .....	18
2.4. Yêu cầu đối với sản phẩm .....	19
2.5. Thiết kế đặc tả hướng đối tượng .....	19
2.5.1. Biểu đồ Use-case tổng quan.....	19
<b>CHƯƠNG 3: XÂY DỰNG GAME TRÊN DESKTOP.....</b>	<b>22</b>
3.1. Xây dựng bản đồ (Map) .....	22
3.1.1 Tài nguyên sử dụng .....	22
3.1.2 Nhân vật .....	24
3.2. Xây dựng chương trình.....	25
3.2.1. Class CameraRotation.....	26
3.2.2. Class DoorAnimtion.....	27
3.2.3. Class FpsGun.....	28
3.2.4. Class TpsGun .....	29
3.2.5. Class IKControl.....	30
3.2.6. Class NameTag .....	31
3.2.7. Class NetworkManager.....	32
3.2.8. Class ImpactLifeCycle .....	34
3.2.9. Class PlayerHealth .....	35

3.2.10. Class PlayerNetworkMover .....	36
3.3 Kiểm thử chương trình .....	37
<b>CHƯƠNG 4: KẾT LUẬN .....</b>	<b>38</b>
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>40</b>

## DANH MỤC HÌNH ẢNH

Hình 1: Thiết lập trên trang chủ của photonengine.....	14
Hình 2: Thêm photonengine vào unity .....	14
Hình 3: Thiết lập các thông số cho photonengine.....	14
Hình 4: Biểu đồ Use-case tổng quan.....	19
Hình 5: Mô hình Buildings .....	22
Hình 6: Các khối Boxes .....	23
Hình 7: Các khối Glasses .....	23
Hình 8: Các khối Blocks .....	23
Hình 9: Mô hình Warehouse .....	23
Hình 10: Toàn bản đồ sau khi thiết kế .....	24
Hình 11: Nhân vật Policeman .....	24
Hình 12: Nhân vật X_Bot .....	24
Hình 13: Nhân vật Y_Bot .....	25
Hình 14: Nhân vật Ethan.....	25
Hình 15: Mẫu súng AK-47.....	25
Hình 16: Class CameraRotation.....	26
Hình 17: Class DoorAnimtion .....	27
Hình 18: Class FpsGun .....	28
Hình 19: Class TpsGun .....	29
Hình 20: Class IKControl.....	30
Hình 21: Class NameTag .....	31
Hình 22: Class NetworkManager.....	32
Hình 23: Class ImpactLifeCycle .....	34
Hình 24: Class PlayerHealth .....	35
Hình 25: Class PlayerNetworkMover .....	36
Hình 26: Kết quả sau khi chơi thử .....	37



## LỜI NÓI ĐẦU

Trong thời đại kỹ thuật số ngày nay, trò chơi điện tử đã trở thành một phần không thể thiếu của cuộc sống hàng ngày của chúng ta. Thể loại trò chơi bắn súng 3D nhiều người chơi đã trở thành một trong những thể loại phổ biến và hấp dẫn nhất trong ngành công nghiệp trò chơi. Đây là những trò chơi không chỉ mang lại trải nghiệm giải trí hấp dẫn, mà còn tạo ra cơ hội giao tiếp và cạnh tranh giữa các người chơi từ khắp nơi trên thế giới.

Với sự phát triển nhanh chóng của công nghệ, việc sử dụng Unity Engine để phát triển trò chơi bắn súng 3D nhiều người chơi đã trở nên phổ biến. Unity Engine cung cấp một nền tảng mạnh mẽ và linh hoạt, cho phép nhà phát triển tạo ra những trò chơi đa người chơi chất lượng cao và tương tác đa dạng.

Trong bài báo cáo “*Game shooter 3D unity multiplayer*” sẽ tìm hiểu về các yếu tố thiết kế, cách thức xây dựng trò chơi, hệ thống multiplayer, và cũng như ảnh hưởng của trò chơi này đến trải nghiệm người chơi. Đồng thời đề xuất các cải tiến và hướng phát triển trong tương lai.

# CHƯƠNG 1: TỔNG QUAN

## 1.1. Giới thiệu về đề tài

Trong thời đại công nghệ số hiện nay, ngành công nghiệp game đang trở thành một trong những ngành phát triển nhanh nhất trên thế giới. Với sự phát triển của công nghệ, các nhà phát triển game đang có nhiều cơ hội để tạo ra những trò chơi ấn tượng với đồ họa chất lượng cao, hình ảnh sống động và âm thanh chân thực.

Trong số các thể loại game phổ biến hiện nay, trò chơi FPS (Multiplayer) đang nhận được sự quan tâm đặc biệt từ người chơi trên toàn thế giới. Đây là loại game đòi hỏi người chơi phải nhanh nhẹn, có khả năng phản xạ cao và tập trung để hoàn thành các nhiệm vụ trong trò chơi. Với tính năng đa người chơi, các trò chơi FPS Multiplayer đã trở thành một xu hướng mới trong ngành công nghiệp game.

Trong đề tài, chủ yếu trình bày về quá trình phát triển một trò chơi FPS Multiplayer bằng công cụ Unity. Dự án sẽ tập trung vào việc thiết kế các tính năng đa người chơi, xử lý mạng và tối ưu hóa game để đảm bảo trò chơi chạy mượt trên các nền tảng khác nhau.

## 1.2. Tổng quan về Unity.

### 1.2.1. Đối tượng tham gia hệ thống

Unity là một engine game đa nền tảng được sử dụng rộng rãi trên toàn thế giới. Nó được phát triển bởi Unity Technologies và được ra mắt vào năm 2005. Unity cho phép các nhà phát triển game tạo ra các trò chơi 2D, 3D và AR/VR cho nhiều nền tảng như PC, Mac, iOS, Android, Xbox, PlayStation và nhiều hơn nữa.

Unity có giao diện đồ họa đơn giản và dễ sử dụng, cho phép người dùng tạo ra các đối tượng, vật liệu và ánh sáng một cách dễ dàng. Nó cũng hỗ trợ các ngôn ngữ lập trình phổ biến như C#, JavaScript và Boo, giúp các nhà phát triển dễ dàng viết mã để tạo ra các tính năng phức tạp cho game.

Một trong những đặc điểm nổi bật của Unity là khả năng xử lý đồ họa và âm thanh. Nó hỗ trợ các công nghệ hiện đại như Shader Graph, Lighting, Audio, Particle System, giúp tạo ra các hiệu ứng đồ họa và âm thanh tuyệt vời.

Unity cũng hỗ trợ các tính năng phát triển game đa người chơi (multiplayer) thông qua các mô-đun phát triển game đa người chơi như Photon, UNet, ....

Với cộng đồng đông đảo và nhiều tài liệu hướng dẫn phong phú, Unity trở thành một trong những engine game phổ biến nhất hiện nay. Nó cũng được sử dụng trong nhiều ứng dụng không chỉ là game, chẳng hạn như các ứng dụng AR/VR, ứng dụng thương mại điện tử, giáo dục...

### *1.2.2. Lịch sử của Unity*

Ngày nay, con người dành khá nhiều thời gian giải trí bên những chiếc smartphone cùng những tựa game yêu thích. Trong số đó có không ít trò chơi được lập trình dựa trên engine Unity 3D đã ra đời cách đây hơn một thập kỉ. Trải qua thời gian phát triển lâu dài và luôn update công nghệ mới, giờ đây Unity 3D đã trở thành lựa chọn số 1 cho bất cứ lập trình viên nào muốn xây dựng một tựa game có thể sử dụng đa nền tảng, chi phí rẻ và dễ thao tác. Tuy rất phổ biến những thực tế ít ai biết được nguồn gốc và lịch sử phát triển của engine.

Unity được phát triển bởi công ty Unity Technologies, thành lập bởi David Helgason, Joachim Ante và Nicholas Francis tại Copenhagen, Đan Mạch vào năm 2004. Ban đầu, Unity được thiết kế để phát triển game trên nền tảng Mac OS X, nhưng sau đó đã được mở rộng để hỗ trợ Windows và nền tảng di động.

Phiên bản đầu tiên của Unity, gọi là Unity 1.0, được ra mắt vào tháng 6 năm 2005. Nó chỉ hỗ trợ phát triển game 3D trên máy tính Mac OS X và được sử dụng chủ yếu cho các trò chơi trên web.

Sau đó, Unity đã phát triển và mở rộng để hỗ trợ nhiều nền tảng hơn, bao gồm Windows, iOS, Android và các nền tảng game console như Xbox và PlayStation. Unity cũng được mở rộng để hỗ trợ phát triển game 2D, AR/VR và các ứng dụng không phải game.

Năm 2012, Unity công bố phiên bản miễn phí của nó, Unity Free, để thu hút các nhà phát triển game indie và người mới bắt đầu trong lĩnh vực phát triển game.

Phiên bản miễn phí đã giúp Unity trở thành một trong những engine game phổ biến nhất hiện nay.

### **1.3. Tổng quan về Photonengine**

Photon Engine là một nền tảng mạng đám mây phát triển trò chơi đa người chơi, được cung cấp bởi Exit Games. Nền tảng này cung cấp một loạt các dịch vụ và công nghệ mạnh mẽ để xây dựng và quản game shooter 3D Unity multiplayer.

Photon Engine được xây dựng dựa trên kiến trúc client-server, trong đó người chơi kết nối và tương tác thông qua một máy chủ trung tâm. Dịch vụ này cung cấp các tính năng quan trọng như kết nối mạng, đồng bộ hóa dữ liệu, xử lý tương tác và quản lý kết nối giữa các người chơi.

Có hai phiên bản chính của Photon Engine:

- Photon Realtime: Đây là phiên bản cơ bản của Photon Engine, tập trung vào việc cung cấp các chức năng cơ bản để kết nối và tương tác giữa người chơi. Nó cung cấp các tính năng như đồng bộ hóa vị trí và trạng thái, xử lý va chạm, chat và phát hiện các người chơi khác trong phòng chơi.
- Photon PUN (Photon Unity Networking): Là một plugin dành riêng cho Unity, Photon PUN giúp tích hợp và sử dụng Photon Engine trong việc phát triển trò chơi shooter 3D Unity multiplayer một cách dễ dàng. Nó cung cấp các API và công cụ giúp tạo ra một mô hình multiplayer linh hoạt và tương tác giữa các người chơi một cách trực quan.

Photon Engine cho phép quản lý một số lượng lớn người chơi cùng lúc, điều chỉnh hiệu suất mạng và độ trễ để đảm bảo trải nghiệm multiplayer mượt mà và ổn định. Nó cũng hỗ trợ các tính năng như lưu trữ và chia sẻ dữ liệu, xếp hạng và ghi điểm, và hệ thống cấu hình phòng chơi.

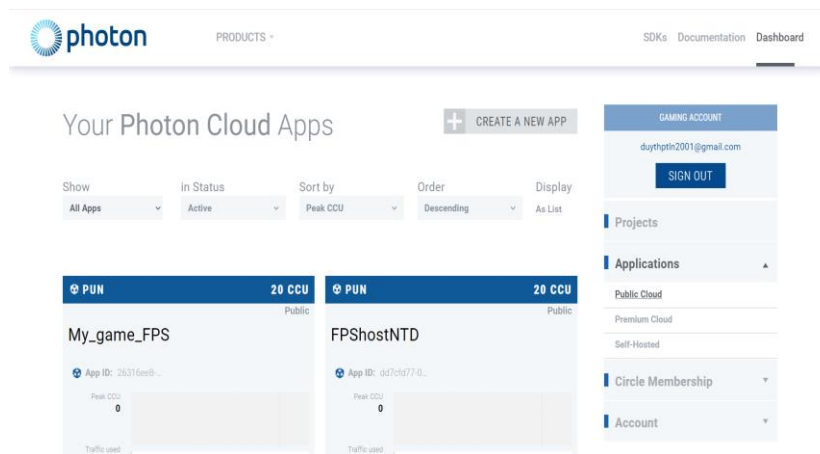
Sử dụng Photon Engine trong việc phát triển trò chơi shooter 3D Unity multiplayer giúp giảm bớt công việc phát triển mạng phức tạp và tập trung vào việc xây dựng gameplay và trải nghiệm người chơi. Nền tảng này đã được sử dụng

rộng rãi và có cộng đồng lớn, đồng nghĩa với việc có nhiều tài liệu hỗ trợ và sự hỗ trợ.

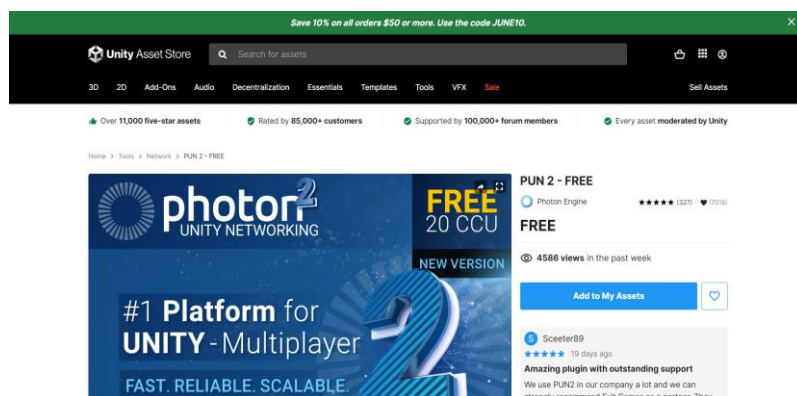
Ngoài Photon, còn có nhiều dịch vụ mạng khác như Mirror, Forge Networking, và UNet (Unity Networking). Các dịch vụ này cung cấp các giải pháp mạng cho việc phát triển trò chơi multiplayer và có thể được sử dụng thay thế cho Photon.

Việc cài đặt Photon Engine là dễ dàng sau đây là một số bước cài đặt:

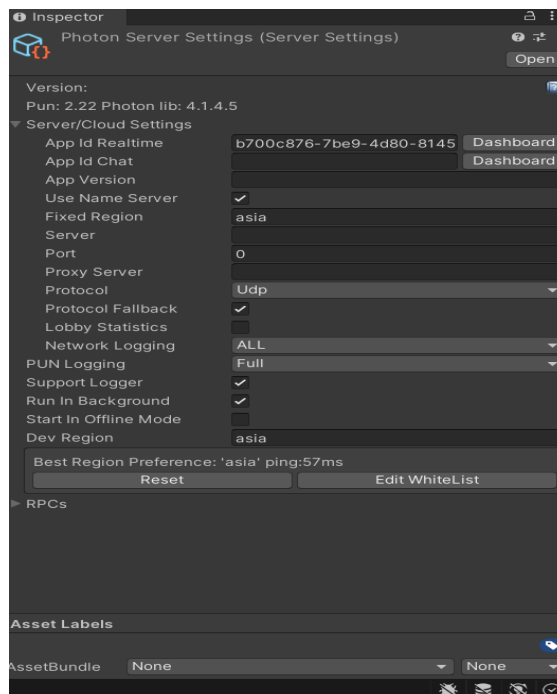
- Đăng ký tài khoản Photon: Truy cập trang web của Photon Engine và đăng ký tài khoản. Có hai phiên bản chính là Photon Realtime và Photon PUN (Photon Unity Networking), bạn có thể chọn phiên bản phù hợp với nhu cầu của bạn.
- Tạo ứng dụng: Sau khi đăng ký và đăng nhập vào tài khoản Photon của bạn, bạn cần tạo một ứng dụng mới. Điều này sẽ cung cấp cho bạn các thông tin cần thiết để tích hợp Photon vào trò chơi của bạn.
- Tích hợp SDK vào dự án Unity: Mở dự án Unity của bạn và import SDK của Photon vào dự án. Điều này sẽ thêm các tệp và thư viện cần thiết để sử dụng Photon trong trò chơi của bạn.
- Cấu hình kết nối: Sử dụng thông tin ứng dụng mà bạn đã tạo trước đó, bạn cần cấu hình kết nối Photon trong mã nguồn của trò chơi của bạn. Điều này bao gồm cung cấp App ID và các thông số kết nối khác để thiết lập kết nối đến máy chủ Photon.



**Hình 1: Thiết lập trên trang chủ của photonengine**



**Hình 2: Thêm photonengine vào unity**



**Hình 3: Thiết lập các thông số cho photonengine**

#### 1.4. Game FPS Multiplayer

Game FPS (First-Person Shooter) Multiplayer là dạng game bắn súng góc nhìn thứ nhất với tính năng chơi đa người chơi. Trong game, người chơi sẽ vào vai một nhân vật và chiến đấu với những người chơi khác để giành chiến thắng. Đây là dạng game rất phổ biến trong cộng đồng game thủ, đặc biệt là trong các giải đấu eSports. Tính năng chơi đa người chơi trong game FPS Multiplayer cho phép người chơi tương tác với những người chơi khác, đồng thời tăng tính thử thách và cạnh tranh của game. Ngoài ra, game FPS Multiplayer cũng có thể tích hợp các tính năng khác như chat, kết nối mạng xã hội, hệ thống mua sắm, tùy chỉnh nhân vật và nhiều tính năng khác để tạo ra trải nghiệm chơi game tốt nhất cho người chơi.

Trong game, góc nhìn là cách mà người chơi quan sát và tương tác với môi trường và các đối tượng trong game. Có ba loại góc nhìn chính được sử dụng trong game:

- Góc nhìn người thứ nhất (First-person view - FPS): trong đó người chơi nhìn từ góc nhìn của nhân vật mà họ đang điều khiển. Trong góc nhìn này, người chơi sẽ thấy toàn bộ môi trường qua mắt của nhân vật. Góc nhìn FPS được sử dụng chủ yếu trong các game bắn súng, game hành động và game phiêu lưu. (FPSGun)
- Góc nhìn thứ ba (Third-person view): trong đó người chơi nhìn nhân vật mà họ đang điều khiển từ phía sau hoặc từ bên cạnh. Góc nhìn thứ ba thường được sử dụng trong các game hành động, game nhập vai và game chiến thuật. (TPSGun)
- Góc nhìn trên cao (Top-down view): trong đó người chơi nhìn từ trên xuống và quan sát toàn bộ môi trường và nhân vật. Góc nhìn này thường được sử dụng trong các game chiến thuật và game xây dựng.

## CHƯƠNG 2: PHÂN TÍCH VÀ THIẾT KẾ

### 2.1. Khảo sát hệ thống

Unity 3D là một công nghệ phát triển game đa nền tảng được sử dụng rộng rãi, bao gồm cả trong thể loại game FPS (First Person Shooter). Dưới đây là một số game FPS đầu tiên được phát triển bằng Unity 3D:

- Dead Trigger (2012): Là một trò chơi FPS zombie đầu tiên được phát triển bằng Unity 3D, Dead Trigger đã được phát hành trên các nền tảng di động và trang Steam.
- Interstellar Marines (2012): Là một trò chơi FPS khoa học viễn tưởng đầu tiên được phát triển bằng Unity 3D, Interstellar Marines đã được phát hành trên Steam và nhận được nhiều đánh giá tích cực.
- Guns of Icarus Online (2012): Là một trò chơi FPS đa người chơi đầu tiên được phát triển bằng Unity 3D, Guns of Icarus Online cho phép người chơi điều khiển các tàu bay trong một thế giới steampunk.
- Strike Vector (2014): Là một trò chơi FPS phi công đầu tiên được phát triển bằng Unity 3D, Strike Vector cho phép người chơi lái các máy bay tàng hình và tham gia vào các trận đấu không gian đầy kịch tính.
- Rust (2013): Là một trò chơi FPS sinh tồn đầu tiên được phát triển bằng Unity 3D, Rust cho phép người chơi tham gia vào một thế giới mở đầy nguy hiểm và cố gắng sống sót trong môi trường đầy thử thách.

Đây chỉ là một số game FPS đầu tiên được phát triển bằng Unity 3D, với sự phát triển của công nghệ, hiện nay đã có rất nhiều game FPS đa dạng và phong phú được phát triển bằng Unity 3D.

#### Điểm nổi bật của dự án:

- Độ phức tạp: Một dự án FPS Multiplayer game có thể có độ phức tạp cao hơn so với các game FPS đầu tiên được phát triển bằng Unity 3D, vì nó bao gồm nhiều yếu tố phức tạp như đồ họa, lập trình đa người chơi, mạng....
- Đồ họa: Đồ họa của các game FPS đầu tiên phát triển bằng Unity 3D thường không được đẹp mắt và chuyên nghiệp như các game FPS hiện đại, bởi vì



Unity 3D còn chưa được phát triển đầy đủ vào thời điểm đó. Trong khi đó, một dự án FPS Multiplayer game có thể có đồ họa đẹp mắt và chuyên nghiệp hơn, vì Unity 3D đã được phát triển và cải tiến rất nhiều kể từ đó.

- Cách chơi: Các game FPS đầu tiên phát triển bằng Unity 3D thường có cách chơi đơn giản và tập trung vào việc bắn súng, trong khi đó một dự án FPS Multiplayer game có thể có nhiều yếu tố khác như chiến thuật, tương tác đa người chơi....
- Đối tượng người chơi: Các game FPS đầu tiên phát triển bằng Unity 3D thường được thiết kế cho một người chơi, trong khi đó một dự án FPS Multiplayer game có thể hướng đến nhiều người chơi cùng lúc.

## **2.2. Thông tin vào ra của hệ thống.**

### Thông tin vào:

- Người chơi nhập thông tin vào hệ thống bằng cách sử dụng bàn phím, chuột, hoặc điều khiển gamepad để tương tác với trò chơi.
- Thông tin vào bao gồm các lựa chọn của người chơi như đăng nhập, tạo tài khoản, chọn chế độ chơi, lựa chọn nhân vật, vũ khí và phương tiện di chuyển.
- Thông tin vào cũng bao gồm các lệnh và hoạt động của người chơi trong trò chơi, chẳng hạn như di chuyển, nhảy, bắn, tấn công hoặc phòng thủ.

### Thông tin ra:

- Hệ thống game FPS Multiplayer trả về các thông tin ra đến người chơi để hiển thị trên màn hình của họ.
- Thông tin ra bao gồm các hình ảnh đồ họa của trò chơi, âm thanh, nhạc nền, các thông báo hệ thống, các kết quả trận đấu, bảng xếp hạng và các thành tích cá nhân của người chơi.
- Hệ thống cũng truyền tải thông tin đến các máy chủ và các người chơi khác trên mạng để đồng bộ hóa dữ liệu và đảm bảo tính đồng bộ trong trò chơi.

## **2.3. Phân tích game**

### *2.3.1. Cách chơi*

Tạo nhân vật và chọn phòng để play game.

Khi đó người chơi sẽ được kết nối đến với những người chơi khác trên cùng một phòng và cùng trên một sever.

Trong trò chơi, người chơi sẽ điều khiển nhân vật của mình bằng bàn phím và chuột để di chuyển, nhảy, bắn và thực hiện các hành động khác.

Nhiệm vụ của người chơi là tiêu diệt các đối thủ của mình trong môi trường chơi đấu và đạt điểm số cao nhất có thể.

Trò chơi kết thúc khi người chơi chọn chơi lại hoặc thoát ra.

### *2.3.2. Bối cảnh game*

Sau 200 năm khi chiến tranh thế giới lần thứ 4 xảy ra ra, trái đất bị tàn phá nặng nề và con người phải sống chung với những hậu quả của cuộc chiến đẫm máu đó. Tài nguyên trên hành tinh trở nên hiếm hơn bao giờ hết và nhiều thành phố đã bị phá hủy hoàn toàn.

Các nhà khoa học và kỹ sư trên toàn thế giới đã phải nỗ lực để xây dựng lại cơ sở hạ tầng và các khu đô thị mới để đáp ứng nhu cầu sống của con người. Tuy nhiên, việc tìm kiếm tài nguyên và tranh giành chúng đã trở thành một vấn đề căn bản.

Những thành phố lớn được xây dựng với công nghệ tiên tiến, nhưng không đủ tài nguyên để đáp ứng nhu cầu sống của toàn bộ dân cư. Vì vậy, những khu vực đó đã trở thành điểm tranh chấp giữa các thế lực trong xã hội mới này.

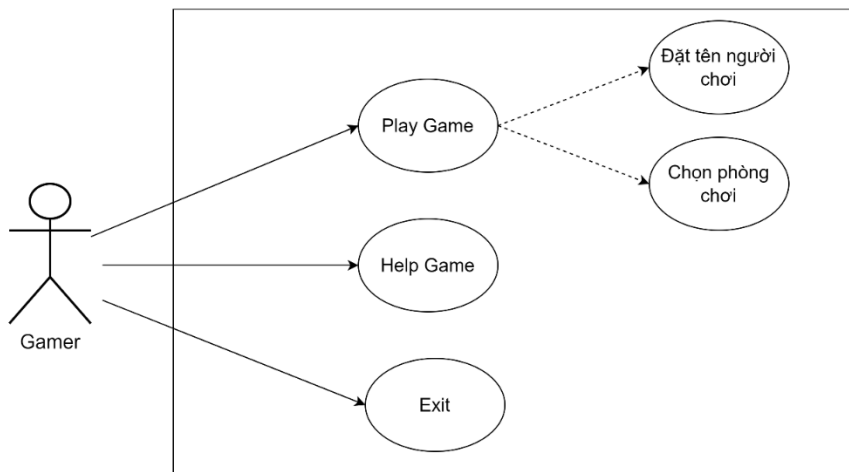
Trong game người chơi sẽ chiến đấu trong một thành phố hoang tàn. Người chơi thực hiện nhiệm vụ của mình để tìm kiếm tài nguyên họ sẽ phải đối mặt với những thử thách khắc nghiệt từ môi trường xung quanh, cũng như các kẻ thù tinh vi và tàn ác. Các trận đấu sẽ diễn ra trên đa dạng các trận địa, từ các thành phố khốc liệt đến các địa hình hiểm trở trên các hành tinh còn đang trong quá trình khám phá.

## 2.4. Yêu cầu đối với sản phẩm

- Game phải có dung lượng không quá cao, tốc độ xử lý nhanh.
- Giao diện trực quan dễ nhìn, thân thiện với người dùng.
- Công việc tính toán chuẩn xác.
- Tạo cảm giác chân thật, hoà sống động với người chơi.
- Âm thanh, hình ảnh hài đặc sắc.
- Phân tích game theo hướng đối tượng cụ thể, rõ ràng.

## 2.5. Thiết kế đặc tả hướng đối tượng

### 2.5.1. Biểu đồ Use-case tổng quan



**Hình 4: Biểu đồ Use-case tổng quan**

#### a) Đặc tả Use-case Play Game

Tên use-case: Use-case Play game.

Người sử dụng: Người dùng.

Mục đích: Chức năng này là chức năng chơi game, là chức năng sử dụng chính của ứng dụng.

Dòng sự kiện:

Hành động của tác nhân	Phản ứng của hệ thống
1. User vào ứng dụng	Hệ thống đưa ra giao diện chính
2. Chọn tên nhân vật và phòng chơi	Hệ thống kết nối đến server với tên người chơi vừa chọn.
3. User nhấn vào button “Play”	Hệ thống hiển thị ra màn hình chơi game.

Các yêu cầu đặc biệt:

- Chọn tên nhân vật và phòng chơi phù hợp.

Trạng thái hệ thống trước khi bắt đầu thực hiện Use-case:

- Hệ thống đang ở màn hình menu game.

Trạng thái hệ thống sau khi thực hiện Use-case:

- Hệ thống hiển thị ra màn hình chơi game chính.

Điểm mở rộng:

- Không có.

*b) Đặc tả Use-case Help Game*

Tên use-case: Use-case Help.

Người sử dụng: Người dùng.

Mục đích: Chức năng giới thiệu về chức năng nhân vật.

Dòng sự kiện:

Hành động của tác nhân	Phản ứng của hệ thống
1. User vào chơi game.	Hệ thống đưa ra giao diện chính.
2. User nhấn vào button “Help”.	Hệ thống hiển thị màn hình giới thiệu các chức năng trong game.

Các yêu cầu đặc biệt:

- Người dùng đã đăng nhập vào game và có kết nối Internet.

Trạng thái hệ thống trước khi bắt đầu thực hiện Use-case:

- Hệ thống đang ở màn hình chính.

Trạng thái hệ thống sau khi thực hiện Use-case:

- Hệ thống hiển thị màn hình giới thiệu các chức năng trong game.

Điểm mở rộng:

- Không có

*c) Đặc tả Use-case Exit Game*

Tên use-case: Use-case Exit.

Người sử dụng: Người dùng.

Mục đích: Người chơi đã bắt đầu trò chơi và muốn thoát khỏi trò chơi hoặc trò chơi đã kết thúc và người chơi đã thoát khỏi trò chơi.

Dòng sự kiện:

Hành động của tác nhân	Phản ứng của hệ thống
1. Người chơi bấm vào nút "Exit" trên màn hình hoặc từ menu trò chơi.	Hệ thống hiển thị hộp thoại xác nhận cho người chơi, yêu cầu người chơi xác nhận việc thoát khỏi trò chơi.
2. Người chơi xác nhận việc thoát khỏi trò chơi.	Hệ thống hiển thị thông báo về việc thoát khỏi trò chơi và đưa người chơi trở lại màn hình chính.

Các yêu cầu đặc biệt:

- Không có

Trạng thái hệ thống trước khi bắt đầu thực hiện Use-case:

- Hệ thống đang ở hoạt động.

Trạng thái hệ thống sau khi thực hiện Use-case:

- Hệ thống hiển thị hộp thoại xác nhận cho người chơi, yêu cầu người chơi xác nhận việc thoát khỏi trò chơi.

Điểm mở rộng:

- Không có

## CHƯƠNG 3: XÂY DỰNG GAME TRÊN DESKTOP

### 3.1. Xây dựng bản đồ (Map)

Xây dựng bản đồ trong trò chơi FPS Multiplayer là một phần quan trọng để tạo ra một môi trường chơi hấp dẫn và thú vị.

Game được xây dựng trong môi trường 3D nên gồm có 3 trục X, Y và Z :

Trục X: Đây là trục chiều ngang hoặc trục chiều rộng. Nó thường được định nghĩa là trục đi từ bên trái sang bên phải. Khi di chuyển dọc theo trục X, các đối tượng thay đổi vị trí theo chiều ngang.

Trục Y: Đây là trục chiều dọc hoặc trục chiều cao. Nó thường được định nghĩa là trục đi từ dưới lên trên. Khi di chuyển dọc theo trục Y, các đối tượng thay đổi vị trí theo chiều dọc.

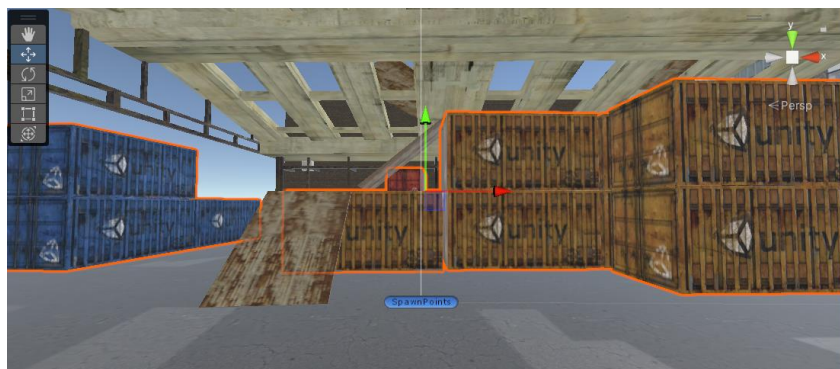
Trục Z: Đây là trục chiều sâu hoặc trục chiều dài. Nó thường được định nghĩa là trục đi từ phía sau về phía trước. Khi di chuyển dọc theo trục Z, các đối tượng thay đổi vị trí theo chiều sâu.

#### 3.1.1 Tài nguyên sử dụng

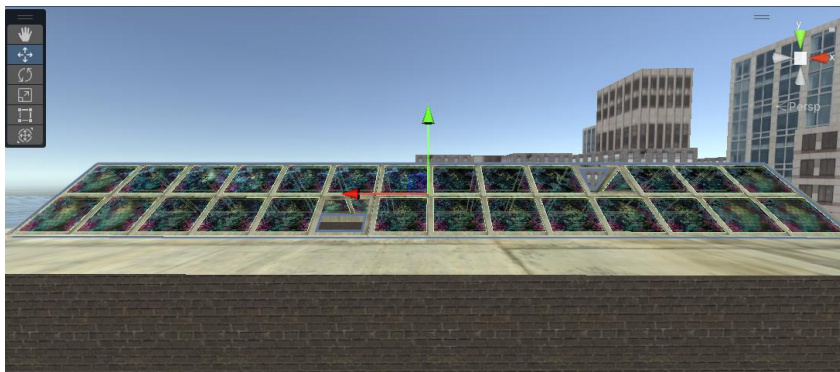
Để xây được một bản đồ để người chơi có thể thao tác và di chuyển cần rất nhiều tài nguyên. Trong bản đồ hiện tại có sử dụng một số tài nguyên như: Air Conditioner (Máy điều hòa), WoodMade (một số vật liệu gỗ), Warehouse (một số thiết kế nhà kho), Doors (một số loại cửa), Pillars (một số các trụ cột) ngoài ra không thể thiếu các Buildings (một số tòa nhà cao tầng), .... Một số hình ảnh minh họa sau:



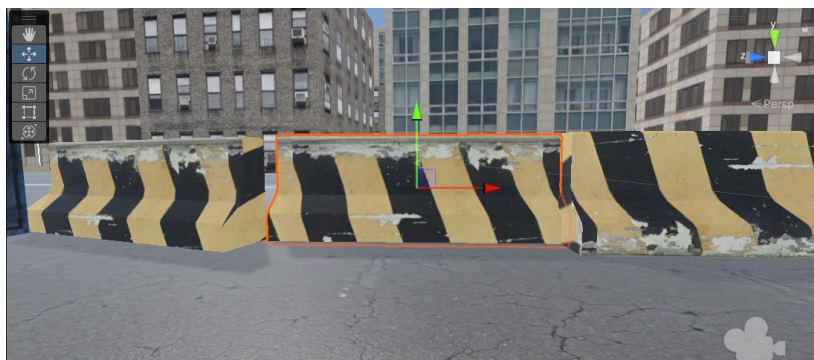
**Hình 5: Mô hình Buildings**



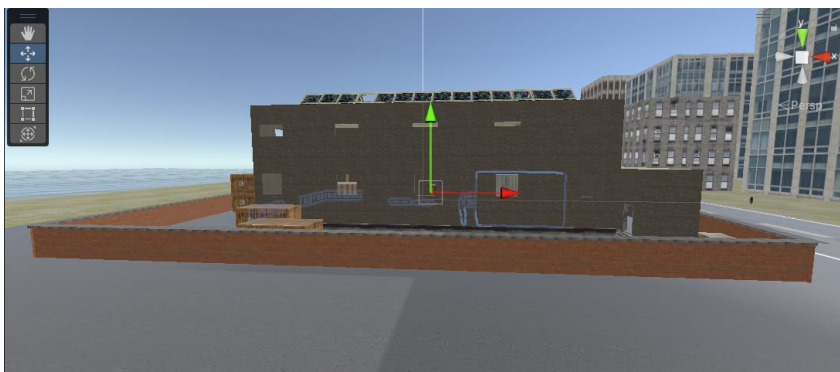
**Hình 6: Các khối Boxes**



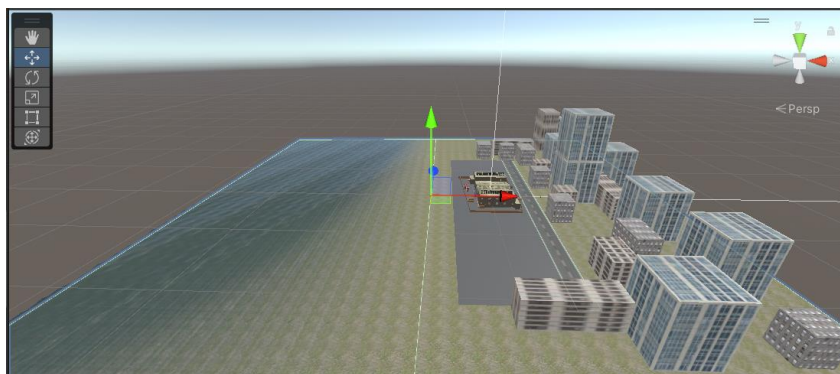
**Hình 7: Các khối Glasses**



**Hình 8: Các khối Blocks**



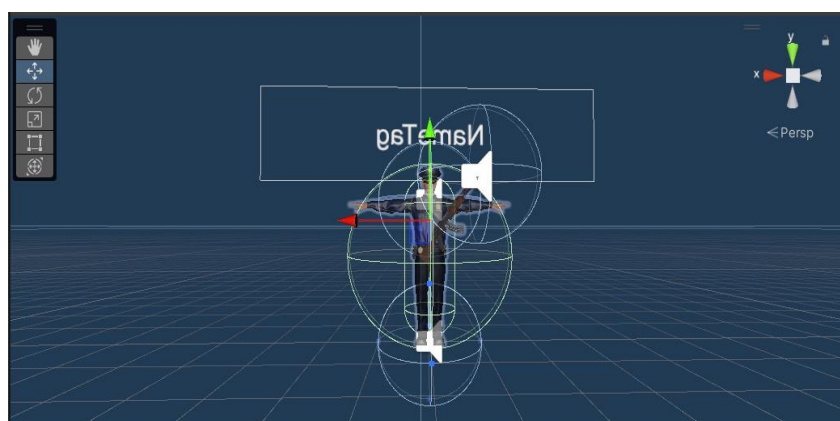
**Hình 9: Mô hình Warehouse**



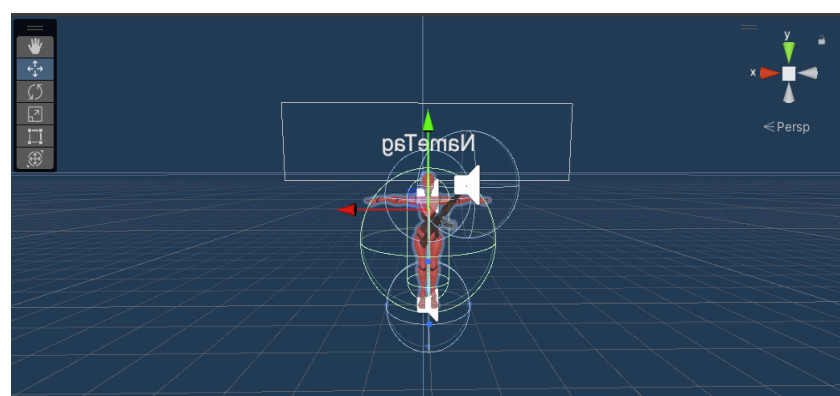
***Hình 10: Toàn bản đồ sau khi thiết kế***

### ***3.1.2 Nhân vật***

Bên cạnh việc xây dựng bản đồ thì việc tạo ra nhân vật chơi cũng rất cần thiết. Sau đây là một số nhân vật được sử dụng.

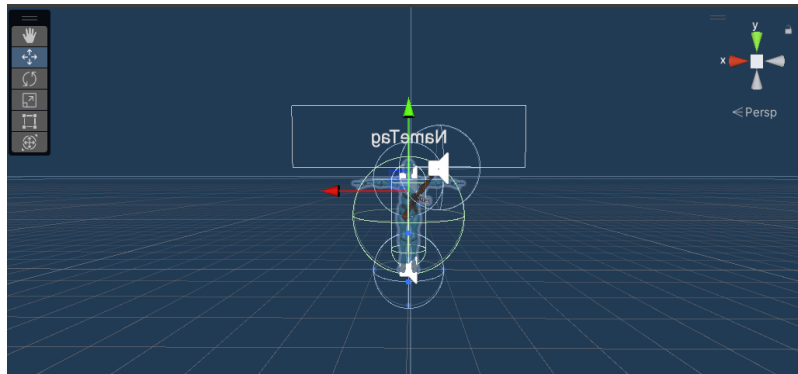


***Hình 11: Nhân vật Policeman***

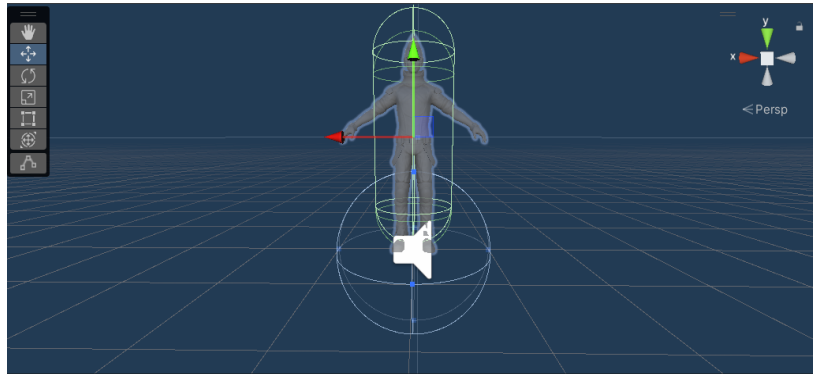


***Hình 12: Nhân vật X\_Bot***

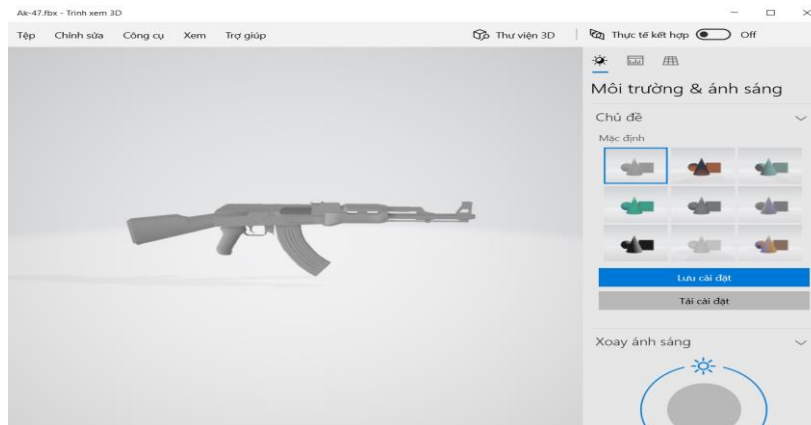




**Hình 13: Nhân vật Y\_Bot**



**Hình 14: Nhân vật Ethan**



**Hình 15: Mẫu súng AK-47**

### 3.2. Xây dựng chương trình

Việc xây dựng chương trình được thực hiện trên IDE Microsoft Visual Studio và có sử dụng Unity tool hỗ trợ code. Gồm một số các class chính:

**Class CameraRotation:** Xoay camera cảnh trong mọi khung hình được cập nhật.

**Class Animtion:** Điều khiển hoạt hình cửa và phát hiện nếu người chơi đi vào hoặc ra khỏi khu vực kích hoạt cửa.

Class FpsGun: Điều khiển súng ở góc nhìn người thứ nhất, chủ yếu để bắn

Class TpsGun: Điều khiển súng ở góc nhìn người thứ ba.

Class IKControl: Đảm bảo mô hình đang cầm súng bất kể chuyển động hay xoay

Class ImpactLifeCycle: Phá hủy đối tượng viên đạn sau vài giây để tiết kiệm thời gian và bộ nhớ của CPU...

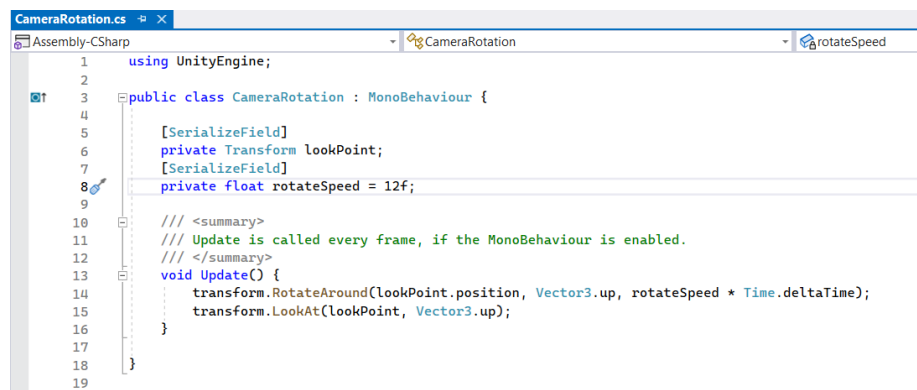
Class NameTag: Hiển thị tên người chơi khác trên đầu họ.

Class NetworkManager: Kiểm soát toàn bộ kết nối mạng

Class PlayerHealth: Tính toán và cập nhật điểm sức khỏe của từng người chơi.

Class PlayerNetworkMover: Đồng bộ hóa vị trí của người chơi giữa các khách hàng khác nhau.

### 3.2.1. Class CameraRotation



```
1 using UnityEngine;
2
3 public class CameraRotation : MonoBehaviour {
4
5     [SerializeField]
6     private Transform lookPoint;
7     [SerializeField]
8     private float rotateSpeed = 12f;
9
10    /// <summary>
11    /// Update is called every frame, if the MonoBehaviour is enabled.
12    /// </summary>
13    void Update() {
14        transform.RotateAround(lookPoint.position, Vector3.up, rotateSpeed * Time.deltaTime);
15        transform.LookAt(lookPoint, Vector3.up);
16    }
17 }
18
19
```

**Hình 16: Class CameraRotation**

Class CameraRotation dùng để điều khiển quá trình quay camera xung quanh một điểm nhìn và luôn nhìn về hướng điểm nhìn đó.

Các thành phần chính là:

Biến lookPoint là một Transform được sử dụng để xác định điểm nhìn cho camera. Điểm nhìn này có thể là một đối tượng trong scene.

Biến rotateSpeed xác định tốc độ quay của camera quanh điểm nhìn. Giá trị này được đặt mặc định là 12f, nhưng có thể điều chỉnh trong Unity Editor.

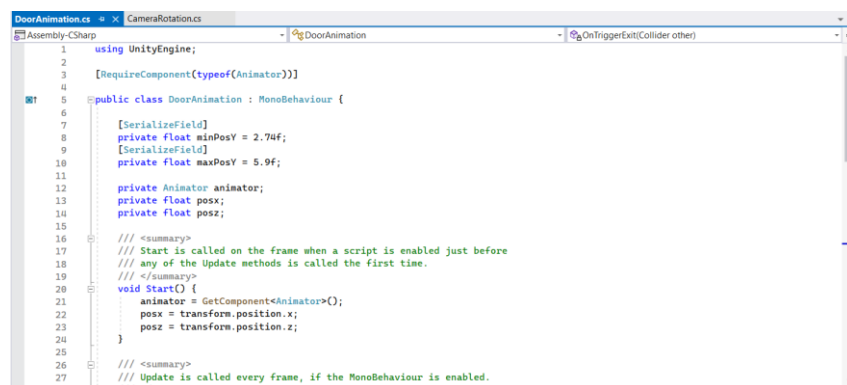
Trong hàm Update(), quá trình quay camera được thực hiện bằng cách sử dụng phương thức RotateAround của thành phần Transform của camera. Hàm này

nhận ba tham số: điểm nhìn, trục quay (ở đây là Vector3.up để quay quanh trục y), và tốc độ quay (nhân với Time.deltaTime để đảm bảo tốc độ không phụ thuộc vào tốc độ khung hình).

Sau đó, phương thức LookAt được sử dụng để camera luôn nhìn về hướng điểm nhìn lookPoint, với Vector3.up xác định hướng lên.

Do script được gắn vào camera, nên nó sẽ được gọi và cập nhật mỗi khung hình.

### 3.2.2. Class DoorAnimtion



```
1 using UnityEngine;
2
3 [RequireComponent(typeof(Animator))]
4
5 public class DoorAnimation : MonoBehaviour {
6
7     [SerializeField]
8     private float minPosY = 2.74f;
9     [SerializeField]
10    private float maxPosY = 5.9f;
11
12    private Animator animator;
13    private float posX;
14    private float posz;
15
16    /// <summary>
17    /// Start is called on the frame when a script is enabled just before
18    /// any of the Update methods is called the first time.
19    /// </summary>
20    void Start() {
21        animator = GetComponent<Animator>();
22        posX = transform.position.x;
23        posz = transform.position.z;
24    }
25
26    /// <summary>
27    /// Update is called every frame, if the MonoBehaviour is enabled.
```

**Hình 17: Class DoorAnimtion**

Class DoorAnimation điều khiển hoạt động mở cửa của một cánh cửa khi người chơi tiếp xúc với nó.

Các thành phần chính là:

- Biến minPosY và maxPosY xác định giới hạn độ cao tối thiểu và tối đa mà cửa có thể di chuyển lên và xuống. Điều này giúp cánh cửa chỉ di chuyển trong một khoảng giới hạn nhất định.
- Biến animator là một tham chiếu tới Animator component của cánh cửa, được sử dụng để điều khiển các trạng thái và hoạt động của animator.
- Biến posX và posz lưu trữ vị trí x và z ban đầu của cánh cửa để giữ cố định vị trí ngang của cửa khi di chuyển theo trục y.

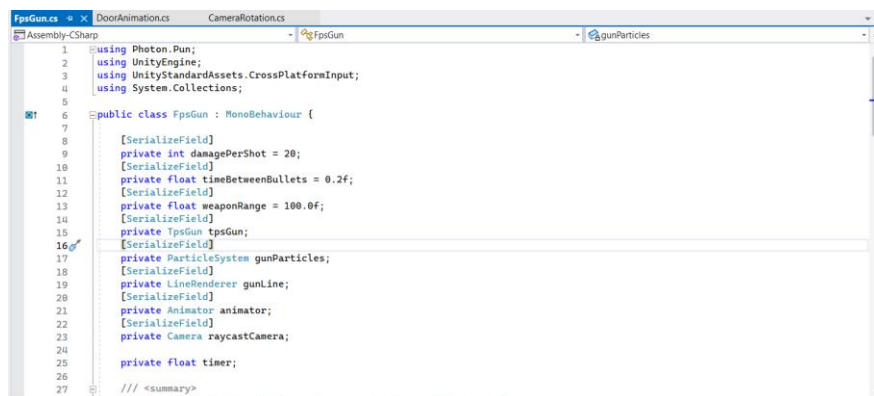
Trong hàm Start(), script lấy tham chiếu tới Animator component và lưu trữ vị trí x và z ban đầu của cửa.

Trong hàm Update(), vị trí của cánh cửa được cập nhật dựa trên vị trí hiện tại của cánh cửa, nhưng với giới hạn độ cao giữa minPosY và maxPosY. Điều này đảm bảo rằng cánh cửa chỉ di chuyển trong khoảng giới hạn đã xác định.

Hàm OnTriggerStay(Collider other) được gọi khi một Collider khác (ở đây là người chơi) tiếp xúc với trigger của cánh cửa. Nếu đối tượng có tag là "Player", biến trạng thái "Trigger" của animator được đặt thành true. Điều này kích hoạt hoạt động mở cửa.

Hàm OnTriggerExit(Collider other) được gọi khi Collider khác không còn tiếp xúc với trigger của cánh cửa. Nếu đối tượng có tag là "Player", biến trạng thái "Trigger" của animator được đặt thành false. Điều này kích hoạt hoạt động đóng cửa.

### 3.2.3. Class FpsGun



```
1 using Photon.Pun;
2 using UnityEngine;
3 using UnityStandardAssets.CrossPlatformInput;
4 using System.Collections;
5
6 public class FpsGun : MonoBehaviour {
7
8     [SerializeField]
9     private int damagePerShot = 20;
10    [SerializeField]
11    private float timeBetweenBullets = 0.2f;
12    [SerializeField]
13    private float weaponRange = 100.0f;
14    [SerializeField]
15    private FpsGun tpsGun;
16    [SerializeField]
17    private ParticleSystem gunParticles;
18    [SerializeField]
19    private LineRenderer gunLine;
20    [SerializeField]
21    private Animator animator;
22    [SerializeField]
23    private Camera raycastCamera;
24
25    private float timer;
26
27    /// <summary>
```

**Hình 18: Class FpsGun**

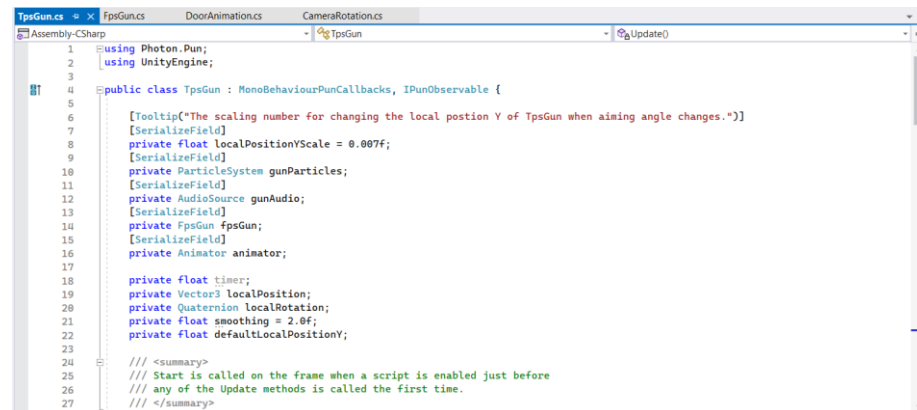
Class FpsGun được sử dụng để điều khiển hoạt động bắn súng của nhân vật người chơi trong góc nhìn thứ nhất (FPS).

Các thành phần chính của script là:

- Biến damagePerShot xác định lượng sát thương mỗi lần bắn.
- Biến timeBetweenBullets xác định thời gian giữa các lần bắn.
- Biến weaponRange xác định khoảng cách tối đa mà đạn có thể đi được.
- Biến tpsGun là một tham chiếu tới script TpsGun để gọi hàm bắn súng cho góc nhìn thứ ba (TPS).

- Biến `gunParticles` và `gunLine` là các `ParticleSystem` và `LineRenderer` được sử dụng để hiển thị hiệu ứng bắn súng.
- Biến `animator` là một tham chiếu tới `Animator` để điều khiển trạng thái bắn súng của nhân vật.
- Biến `raycastCamera` là một tham chiếu tới `Camera` được sử dụng để tạo tia chạy để xác định điểm va chạm.

### 3.2.4. Class *TpsGun*



**Hình 19: Class *TpsGun***

Class `TpsGun` được sử dụng để điều khiển hoạt động bắn súng của nhân vật người chơi trong góc nhìn thứ ba (TPS). Trong class này là sử dụng `Photon` để đồng bộ hóa các hành động bắn súng giữa các người chơi trong mạng.

Các thành phần chính là:

- Biến `localPositionYScale` là một hệ số để điều chỉnh vị trí cục bộ theo trục Y của `TpsGun` khi góc ngắm thay đổi.
- Biến `gunParticles` và `gunAudio` là các `ParticleSystem` và `AudioSource` được sử dụng để hiển thị hiệu ứng và âm thanh bắn súng.
- Biến `fpsGun` là một tham chiếu tới script `FpsGun` để đồng bộ hóa việc bắn súng giữa góc nhìn thứ nhất và thứ ba.
- Biến `animator` là một tham chiếu tới `Animator` để điều khiển trạng thái bắn súng của nhân vật.

Trong hàm `Start()`, script kiểm tra nếu đối tượng đang được điều khiển bởi người chơi hiện tại (`photonView.IsMine`), nó sẽ lưu trữ vị trí mặc định của

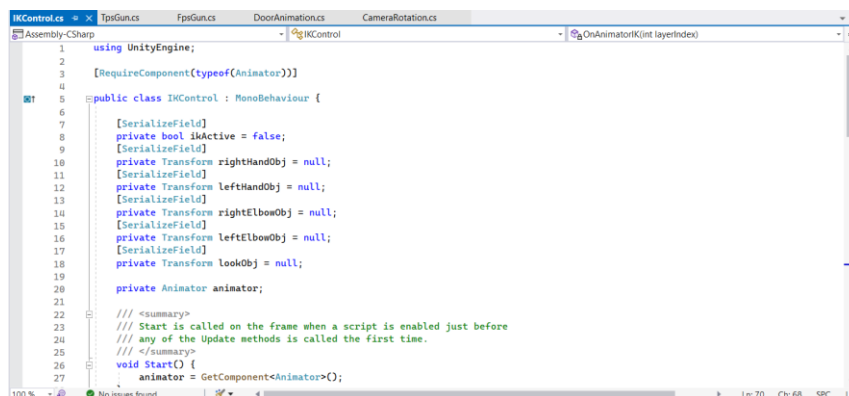
TpsGun. Ngược lại, nếu đối tượng không thuộc sở hữu của người chơi hiện tại, nó sẽ lưu trữ vị trí và hướng quay ban đầu của TpsGun.

Trong hàm Update(), nếu đối tượng thuộc sở hữu của người chơi hiện tại, hướng quay của TpsGun sẽ được đồng bộ với hướng quay của FpsGun.

Trong hàm LateUpdate(), nếu đối tượng thuộc sở hữu của người chơi hiện tại, TpsGun sẽ được điều chỉnh vị trí cục bộ theo góc ngắm bằng cách tính toán sự thay đổi của góc quay theo trục X và áp dụng cho vị trí cục bộ của TpsGun. Ngược lại, nếu đối tượng không thuộc sở hữu của người chơi hiện tại, TpsGun sẽ được cập nhật vị trí và hướng quay thông qua việc sử dụng hàm Lerp để tạo hiệu ứng mượt mà.

Hàm RPCShoot() được sử dụng để gọi RPC (Remote Procedure Call) để bắn súng.

### 3.2.5. Class IKControl



**Hình 20: Class IKControl**

Class IKControl được sử dụng để điều khiển IK (inverse kinematics) trong animator của nhân vật. IKControl cho phép đặt vị trí và quay của các phần tử như tay, khuỷu tay và đầu của nhân vật dựa trên các đối tượng mục tiêu được chỉ định. Hay hiểu đơn giản là điều khiển các điểm trên khung xương nhân vật được thiết lập sẵn.

Các thành phần chính là:

- Biến ikActive xác định xem IK có được kích hoạt hay không.



Class kế thừa từ `MonoBehaviourPunCallbacks`, là một lớp cụ thể của Photon cung cấp các hàm callback cho các sự kiện mạng.

Cách class hoạt động như sau:

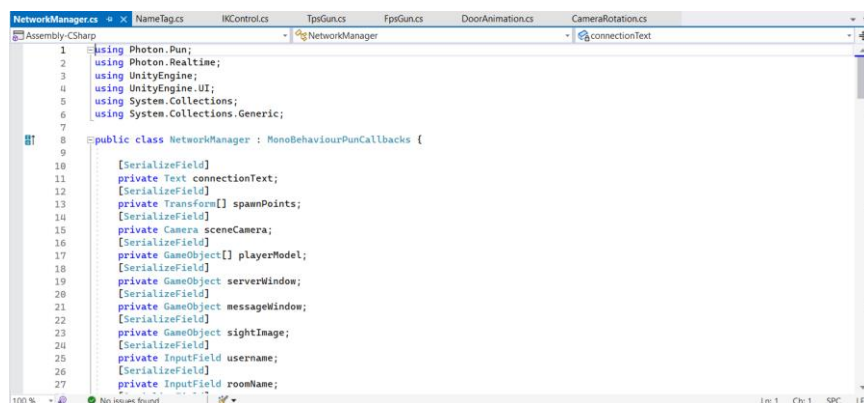
Trong hàm `Start()`, nếu `photonView.IsMine` là đúng (nghĩa là script được chạy trên client của người chơi hiện tại), nó gọi `photonView.RPC` để gửi RPC "SetName" cho tất cả các client khác, truyền vào tên người chơi của `PhotonNetwork.NickName`.

Nếu `photonView.IsMine` là sai (nghĩa là script đang chạy trên client của người chơi khác), nó gọi hàm `SetName` trực tiếp để đặt tên cho `nameText` bằng tên của chủ sở hữu `photonView.Owner.NickName`.

Trong hàm `Update()`, nếu `target` không null, script sẽ điều chỉnh hướng nhìn của `nameTag` để nhìn vào một điểm tính toán dựa trên vị trí của `target`. Điều này giúp `nameTag` luôn hướng về phía nhân vật của người chơi.

Hàm RPC "SetName" được gọi từ xa để đặt tên cho `nameText`. Khi nhận được RPC này, class sẽ đặt giá trị của `nameText` thành tên được truyền vào.

### 3.2.7. Class *NetworkManager*



**Hình 22: Class *NetworkManager***

Class `NetworkManager` được sử dụng để quản lý kết nối và tạo phòng trong trò chơi đa người sử dụng Photon Unity Networking (PUN). Dưới đây là mô tả tổng quan về cách class hoạt động:

Class kế thừa từ `MonoBehaviourPunCallbacks`, là một lớp cụ thể của Photon cung cấp các hàm callback cho các sự kiện mạng.



Có các biến công khai kiểu Text, Transform, Camera và GameObject được gắn kết với các thành phần trong giao diện người dùng để hiển thị thông tin, quản lý vị trí xuất hiện của nhân vật và các đối tượng khác.

Có các biến khác nhau để lưu trữ thông tin như người chơi, hàng đợi tin nhắn, số lượng tin nhắn, và khóa ưu tiên của tên người chơi.

Cách class hoạt động như sau:

Trong hàm Start(), script khởi tạo hàng đợi tin nhắn và kiểm tra nếu đã có tên người chơi được lưu trữ từ trước (sử dụng PlayerPrefs), nó sẽ hiển thị tên đó trong trường nhập liệu tên người dùng.

Khi đã khởi tạo, script kết nối với máy chủ Master sử dụng cài đặt của PhotonNetwork và hiển thị thông báo "Connecting to lobby...".

Hàm callback OnConnectedToMaster() được gọi khi kết nối thành công đến máy chủ Master. Trong hàm này, script gọi PhotonNetwork.JoinLobby() để tham gia vào lobby.

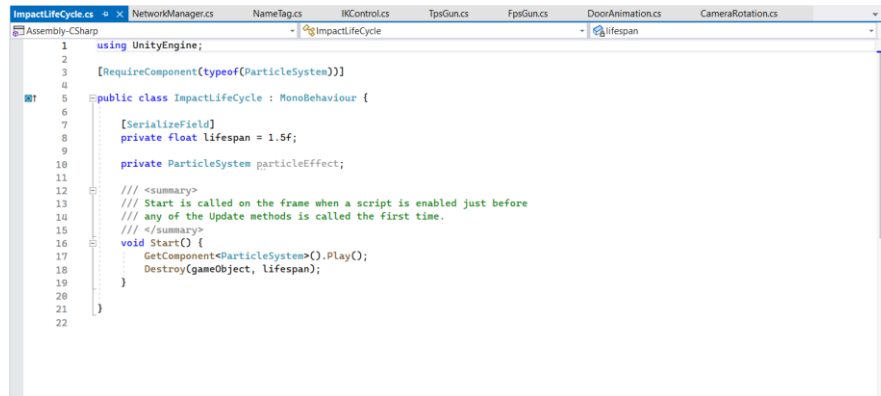
Nếu kết nối bị mất hoặc ngắt kết nối từ máy chủ, hàm callback OnDisconnected() được gọi và hiển thị nguyên nhân ngắt kết nối trên giao diện người dùng.

Hàm callback OnJoinedLobby() được gọi khi người chơi tham gia vào lobby thành công. Trong hàm này, script hiển thị cửa sổ máy chủ và xóa nội dung của trường nhập liệu danh sách phòng.

Hàm callback OnRoomListUpdate() được gọi khi danh sách phòng thay đổi. Trong hàm này, script xóa nội dung của trường nhập liệu danh sách phòng và cập nhật nội dung mới dựa trên danh sách phòng được cung cấp.

Hàm JoinRoom() được gọi khi người chơi nhấp vào nút "Join Room".

### 3.2.8. Class *ImpactLifeCycle*



```
1 using UnityEngine;
2
3 [RequireComponent(typeof(ParticleSystem))]
4
5 public class ImpactLifeCycle : MonoBehaviour {
6
7     [SerializeField]
8     private float lifespan = 1.5f;
9
10    private ParticleSystem particleEffect;
11
12    /// <summary>
13    /// Start is called on the frame when a script is enabled just before
14    /// any of the Update methods is called the first time.
15    /// </summary>
16    void Start() {
17        GetComponent<ParticleSystem>().Play();
18        Destroy(gameObject, lifespan);
19    }
20
21 }
22
```

**Hình 23: Class *ImpactLifeCycle***

Class *ImpactLifeCycle* được sử dụng để xử lý vòng đời của hiệu ứng hạt nhân trong Unity. Hay còn được hiểu là huỷ đi các hiệu ứng khi không cần thiết (VD: như lúc bắn đạn thì sau 5 giây các hiệu ứng xuất hiện trước sẽ tự huỷ) Dưới đây là mô tả tổng quan về cách script hoạt động:

Trong hàm *Start()*, script lấy thành phần *ParticleSystem* được gắn kết với *GameObject* hiện tại và bắt đầu phát hiệu ứng hạt nhân bằng cách gọi *Play()*.

Sau đó, script sẽ gọi hàm *Destroy(gameObject, lifespan)* để huỷ *GameObject* hiện tại sau một khoảng thời gian sống đã được thiết lập.

Cách script hoạt động như sau:

Khi *GameObject* được kích hoạt, hàm *Start()* được gọi.

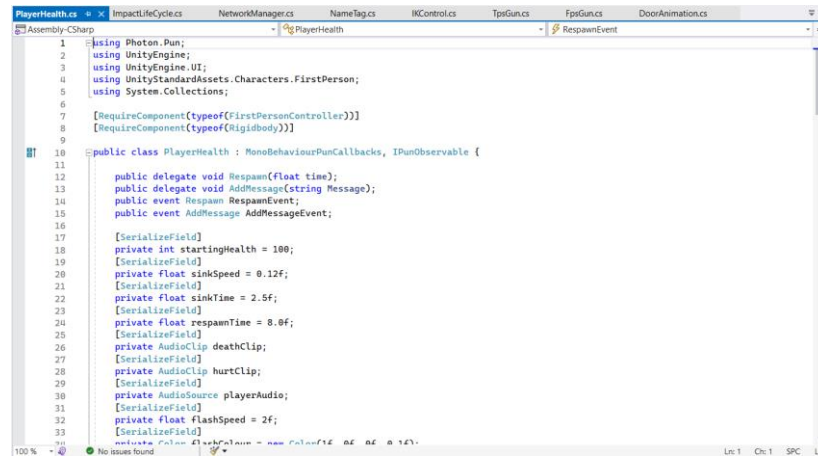
Trong hàm *Start()*, *ParticleSystem* được khởi động bằng cách gọi *Play()*.

Sau đó, script sẽ gọi hàm *Destroy(gameObject, lifespan)* để huỷ *GameObject* hiện tại sau một khoảng thời gian sống đã thiết lập.

Khi thời gian sống đã trôi qua, *GameObject* sẽ bị huỷ và hiệu ứng hạt nhân kết thúc.

Điều này giúp xử lý việc tự động huỷ hiệu ứng hạt nhân sau một khoảng thời gian nhất định, đảm bảo rằng tài nguyên của máy tính được giải phóng sau khi hiệu ứng hoàn thành.

### 3.2.9. Class *PlayerHealth*



**Hình 24: Class *PlayerHealth***

Class *PlayerHealth* được sử dụng để quản lý sức khỏe của người chơi (lượng hp hay máu). Dưới đây là mô tả tổng quan về cách script hoạt động:

Trong hàm *Start()*, script khởi tạo các biến và lấy các thành phần cần thiết từ *GameObject*. Nếu *photonView* là người chơi hiện tại, nó cài đặt giá trị sức khỏe ban đầu và đặt lớp của *GameObject* là "*FPSPlayer*". Script cũng khởi tạo các biến khác như *isDead*, *isSinking*, *damaged*.

Trong hàm *Update()*, script kiểm tra xem người chơi có bị thương hay không và cập nhật màu của hình ảnh hiệu ứng chịu sát thương. Nếu người chơi đang chìm, script sẽ di chuyển *GameObject* xuống dưới.

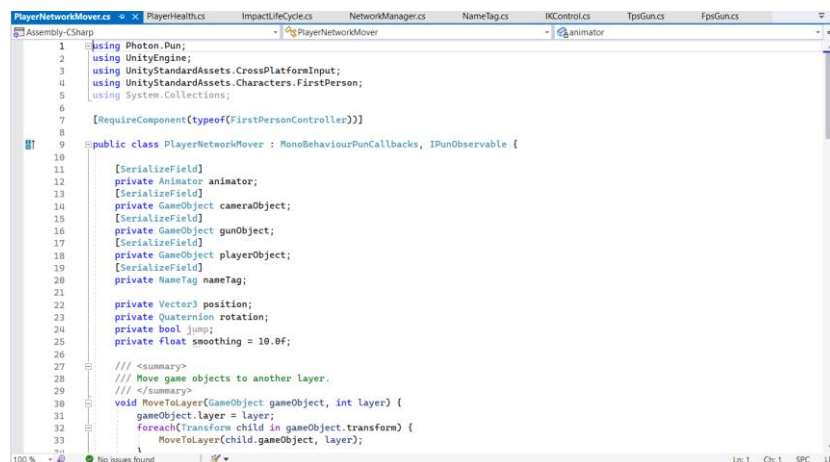
Hàm *TakeDamage(int amount, string enemyName)* được gọi từ một RPC function để gửi thông điệp với số lượng sát thương và tên kẻ địch gây ra sát thương. Nếu người chơi đã chết, hàm sẽ không làm gì cả. Nếu *photonView* là người chơi hiện tại, script sẽ ghi nhận việc bị thương, giảm sức khỏe và kiểm tra nếu sức khỏe nhỏ hơn hoặc bằng 0 thì gọi RPC function *Death()* để xử lý việc chết. Âm thanh bị thương cũng được phát.

Hàm *Death(string enemyName)* được gọi từ một RPC function để xử lý việc chết của người chơi. Nếu *photonView* là người chơi hiện tại, script sẽ ghi nhận trạng thái đã chết, vô hiệu hóa các thành phần, phát âm thanh chết, gọi sự kiện *RespawnEvent*, và khởi động các coroutine để chìm và hủy *GameObject* sau một khoảng thời gian.

Hai coroutine khác được sử dụng để xử lý việc hủy GameObject và chìm nó xuống.

Hàm `OnPhotonSerializeView(PhotonStream stream, PhotonMessageInfo info)` được sử dụng để đồng bộ hóa các biến trong script khi được xem qua mạng Photon. Nếu `stream.IsWriting` là `true`, nghĩa là người chơi hiện tại đang gửi dữ liệu, script sẽ gửi giá trị hiện tại của biến `currentHealth` qua stream. Ngược lại, nếu `stream.IsWriting` là `false`, script sẽ nhận giá trị của biến `currentHealth` từ stream.

### 3.2.10. Class *PlayerNetworkMover*



**Hình 25: Class *PlayerNetworkMover***

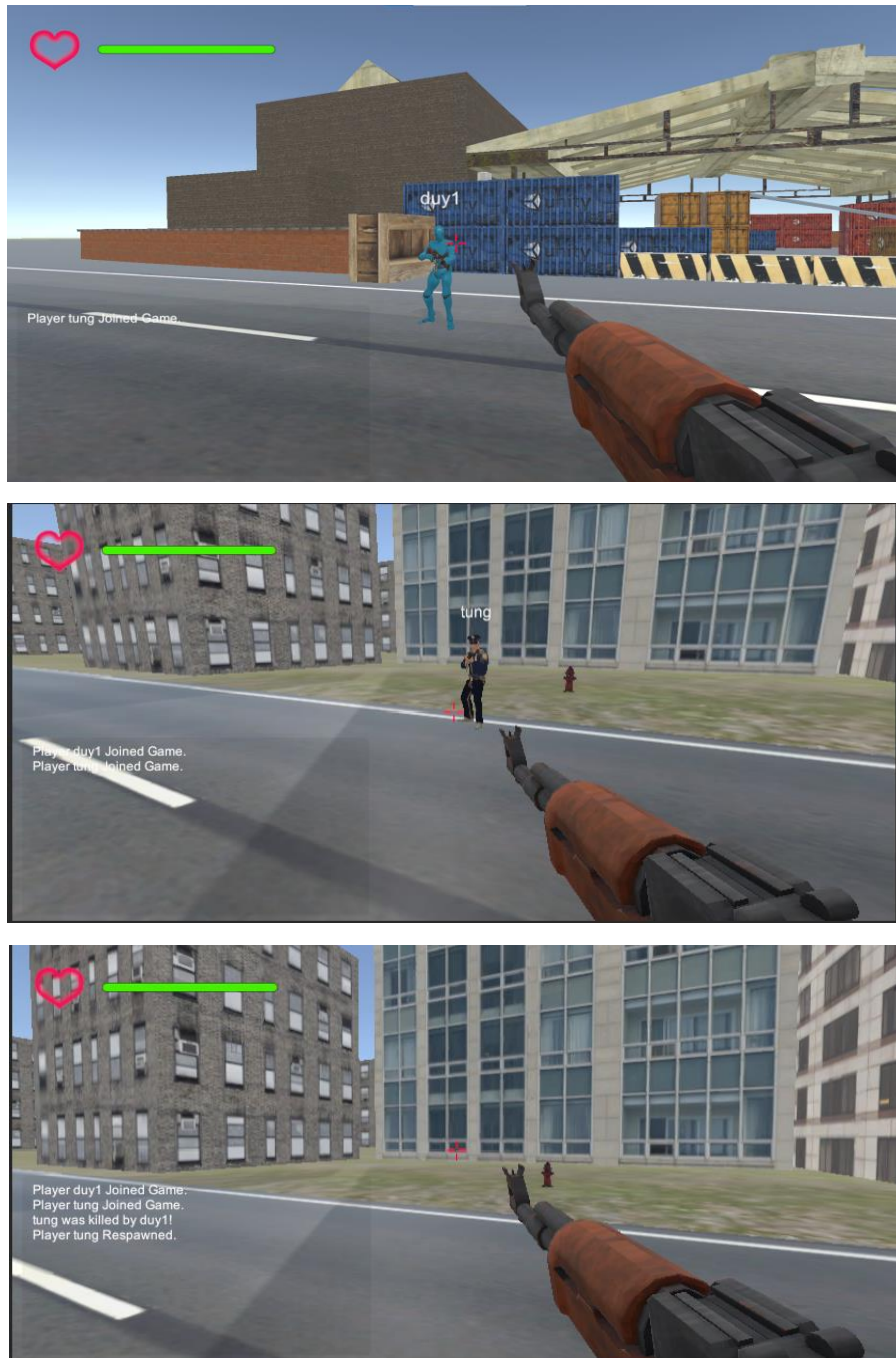
Class *PlayerNetworkMover* này được sử dụng để đồng bộ hóa chuyển động và hoạt ảnh của nhân vật người chơi qua mạng sử dụng Photon, giúp các người chơi trong cùng một phòng chơi có thể nhìn thấy và tương tác với nhau.

Nếu nhân vật đang chạy trên máy chủ của mình (`photonView.IsMine == true`), script cho phép người chơi điều khiển nhân vật và các đối tượng vũ khí và nhân vật của mình sẽ được hiển thị.

Nếu nhân vật đang chạy trên máy khách (không phải máy chủ), script sẽ đồng bộ hóa vị trí và hướng quay của nhân vật từ máy chủ. Class sử dụng phương thức `Lerp` để mịn hóa chuyển động của nhân vật khi nhận dữ liệu từ máy chủ.

Trong phương thức `FixedUpdate`, script sẽ cập nhật trạng thái của animator dựa trên các đầu vào từ người chơi, bao gồm các trục ngang (`Horizontal`) và dọc (`Vertical`), chuyển động nhảy (`IsJumping`) và chạy (`Running`).

### 3.3 Kiểm thử chương trình



*Hình 26: Kết quả sau khi chơi thử*

## CHƯƠNG 4: KẾT LUẬN

Như vậy, việc phát triển “*Game shooter 3D Unity multiplayer*” là một quá trình phức tạp và đòi hỏi sự tập trung vào nhiều khía cạnh. Sử dụng Photon Engine, một nền tảng mạng đám mây phát triển trò chơi đa người chơi, là một lựa chọn thông minh để xây dựng và quản lý chế độ multiplayer của trò chơi. Trong quá trình xây dựng và thiết kế thì sau đây là một số những ưu nhược điểm của đề tài:

### ***Ưu điểm:***

Trải nghiệm chơi game đa người chơi: Người chơi đã có thể kết nối với những người khác để chiến đấu.

Cạnh tranh: Trò chơi multiplayer tạo ra những thách thức mới và cơ hội cạnh tranh giữa các người chơi. Việc chơi với người chơi thực tế mang lại trải nghiệm thú vị và học hỏi từ các chiến thuật và kỹ năng của nhau.

### ***Nhược điểm:***

Phát triển phức tạp: Xây dựng một trò chơi shooter 3D Unity multiplayer đòi hỏi kiến thức về lập trình, đồ họa, mạng và tương tác người chơi. Việc đảm bảo đồng bộ hóa dữ liệu, quản lý kết nối và xử lý va chạm là một quá trình rất khó khăn.

Đòi hỏi tài nguyên: Trò chơi 3D multiplayer yêu cầu tài nguyên lớn, bao gồm mô hình 3D, textures, âm thanh và cơ sở dữ liệu..

Đôi tượng người chơi: Hiện tại số lượng người chơi còn hạn chế tối đa chỉ 20 người chơi.

Trải nghiệm: Quá trình đồng bộ hoá còn chưa được tối ưu dẫn đến hiện tượng giật, lác gậy khó chịu

Kho vũ khí, đồ họa, nhân vật: Còn thiếu rất nhiều các vũ khí

### ***Phương hướng phát triển:***

Mở rộng tính năng và nội dung: Tạo thêm các chế độ chơi mới, bản đồ, vũ khí, và tính năng độc đáo để làm giàu trải nghiệm người chơi.

Tối ưu hóa hiệu suất: Tiếp tục tối ưu hóa hiệu suất của trò chơi để đảm bảo chạy mượt mà và ổn định trên nhiều nền tảng và thiết bị khác nhau.

Tăng cường tương tác người chơi: Phát triển hệ thống chat, hệ thống lời mời chơi, và các tính năng tương tác khác để người chơi có thể giao tiếp và tương tác một cách thuận tiện và thú vị. Đồng thời, tạo ra cơ chế xử lý gian lận và bảo mật để đảm bảo môi trường chơi game công bằng và an toàn.

Đa nền tảng và đa kênh: Nâng cấp trò chơi để hỗ trợ nhiều nền tảng và đa kênh chơi, bao gồm PC, console, di động và VR. Điều này sẽ mở rộng khả năng tiếp cận của trò chơi và thu hút được đa dạng người chơi.

## TÀI LIỆU THAM KHẢO

- [1][Tài liệu sử dụng photon](#)
- [2][Tài liệu làm game multiplayer](#)
- [3][https://www.youtube.com/watch?v=CXGtn4AbH4Q&list=PLzDmz6YkbPZg-Yy-B1\\_44OK7Aoy8L\\_wu-](https://www.youtube.com/watch?v=CXGtn4AbH4Q&list=PLzDmz6YkbPZg-Yy-B1_44OK7Aoy8L_wu-)
- [4]<https://www.youtube.com/watch?v=ZNH6CE--BAA&list=PLzDmz6YkbPZidYcDI5R24PknIuove6X->