TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN TỐT NGHIỆP**

**TÌM HIỂU CÔNG NGHỆ VR   
VÀ XÂY DỰNG ỨNG DỤNG GAME HÀNH ĐỘNG CHẠY TRÊN VR**

*Người hướng dẫn*: **Thầy VŨ ĐÌNH HỒNG**

*Người thực hiện*: **NGUYỄN DUY HÀN LÂM – MSSV: 51403229**

**TRẦN TRUNG THỊNH – MSSV: 51403351**

Khoá  **: 18**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2018**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**ĐỒ ÁN TỐT NGHIỆP**

**TÌM HIỂU CÔNG NGHỆ VR   
VÀ XÂY DỰNG ỨNG DỤNG GAME HÀNH ĐỘNG CHẠY TRÊN VR**

*Người hướng dẫn*: **Thầy VŨ ĐÌNH HỒNG**

*Người thực hiện*: **NGUYỄN DUY HÀN LÂM – MSSV: 51403229**

**TRẦN TRUNG THỊNH – MSSV: 51403351**

Khoá  **: 18**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2018**

LỜI CẢM ƠN

Nhóm chúng em xin chân thành cảm ơn thầy Vũ Đình Hồng vì đã tận tình hướng dẫn chúng em hoàn thành đồ án này.

**CÔNG TRÌNH ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của Thầy Vũ Đình Hồng;. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong luận văn còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung luận văn của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 15 tháng 11 năm 2017*

*Tác giả*

*(ký tên và ghi rõ họ tên)*

*Nguyễn Duy Hàn Lâm*

*Trần Trung Thịnh*

TÓM TẮT

VR (Virtual Reality) hay còn gọi là thực tế ảo là một công nghệ mô phỏng có thể mang lại cho con người một cảm giác gần như thực khi cảm nhận, cảm giác về các bức ảnh, khung cảnh được giả lập, hay được chụp lại. Đồ án này xin trình bày về công nghệ VR là gì, các thiết bị liên quan cũng như các nền tảng VR hiện nay và đồng thời xây dựng ứng dụng (ứng dụng trò chơi) trên nền tảng công nghệ VR này.

MỤC LỤC

[LỜI CẢM ƠN i](#_Toc507759747)

[TÓM TẮT iii](#_Toc507759748)

[MỤC LỤC 1](#_Toc507759749)

[DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT 4](#_Toc507759750)

[DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ 5](#_Toc507759751)

[CHƯƠNG 1 – TỔNG QUAN VỀ VIRTUAL REALITY 9](#_Toc507759752)

[1.1 Khái niệm về Virtual Reality 9](#_Toc507759753)

[1.2 Nguồn gốc của thuật ngữ 10](#_Toc507759754)

[1.3 Công nghệ 10](#_Toc507759755)

[1.4 Nguyên lý hoạt động của VR 11](#_Toc507759756)

[CHƯƠNG 2 – TỔNG QUAN VỀ TRÒ CHƠI 14](#_Toc507759757)

[2.1 Các thông tin chung về trò chơi 14](#_Toc507759758)

[2.1.1 Thông tin về trò chơi 14](#_Toc507759759)

[2.1.2 Bối cảnh của trò chơi 14](#_Toc507759760)

[2.1.3 Mục tiêu hoàn thành của trò chơi 14](#_Toc507759761)

[2.1.4 Các tương tác của người dùng 14](#_Toc507759762)

[2.2 Thông tin về bản đồ, nhân vật và kẻ thù 15](#_Toc507759763)

[2.2.1 Thông tin chung về bản đồ 15](#_Toc507759764)

[2.2.2 Thông tin chung về nhân vật 16](#_Toc507759765)

[2.2.3 Thông tin chung về kẻ thù 17](#_Toc507759766)

[2.3 Chi tiết về nhân vật 17](#_Toc507759767)

[2.3.1 Xây dựng di chuyển cho nhân vật 17](#_Toc507759768)

[2.3.2 Xây dựng hoạt động nhảy của nhân vật 18](#_Toc507759769)

[2.3.3 Tạo khung nhìn cho nhân vật 19](#_Toc507759770)

[2.3.4 Xây dựng hoạt động bắn cho nhân vật 20](#_Toc507759771)

[2.3.5 Xây dựng hoạt động nạp đạn cho nhân vật 22](#_Toc507759772)

[2.4 Chi tiết về kẻ thù 23](#_Toc507759773)

[2.4.1 Cụ thể từng loại zombie 23](#_Toc507759774)

[2.4.2 Xây dựng chuỗi hoạt động cho kẻ thù 28](#_Toc507759775)

[2.4.3 Xây dựng cách kẻ thù tìm người chơi 30](#_Toc507759776)

[2.4.4 Xây dựng cách kẻ thù tấn công người chơi 33](#_Toc507759777)

[2.5 Tổng kết chương 33](#_Toc507759778)

[CHƯƠNG 3 – XÂY DỰNG HỆ ĐA NGƯỜI CHƠI TRÊN NỀN MẠNG NỘI BỘ (LAN) 34](#_Toc507759779)

[3.1 Tổng quan về mạng lưới đa người chơi của Unity 34](#_Toc507759780)

[3.1.1 High Level Scripting API (High Level API hay HLAPI) 35](#_Toc507759781)

[3.1.2 Trình soạn thảo và trình hoạt động tích hợp 37](#_Toc507759782)

[3.1.3 Các dịch vụ Internet của Unity 37](#_Toc507759783)

[3.1.4 Network Transport và Network Layer 37](#_Toc507759784)

[3.2 Nguyên tắc hoạt động cơ bản của mạng đa người chơi nội bộ của Unity 38](#_Toc507759785)

[3.2.1 Quá trình khởi tạo 39](#_Toc507759786)

[3.2.2 Người chơi, người chơi địa phương và sự ủy quyền 44](#_Toc507759787)

[3.2.3 Sự ủy quyền cho các đối tượng không phải là người chơi 47](#_Toc507759788)

[3.2.4 Các tính chất phạm vi của hệ thống mạng 47](#_Toc507759789)

[3.3 Network Manger 48](#_Toc507759790)

[3.4 Thiết lập prefab nhân vật của người chơi 48](#_Toc507759791)

[3.5 Viết Script mạng cho nhân vật 52](#_Toc507759792)

[3.5.1 Chuyển từ script thông thường sang script mạng 52](#_Toc507759793)

[3.5.2 Lọc bỏ các thành phần không thuộc người chơi 52](#_Toc507759794)

[3.6 Quản lý vật thể ngoài phạm vi của người chơi trong hệ thống mạng 53](#_Toc507759795)

[3.6.1 Các hàm quản lý trạng thái trực tuyến 53](#_Toc507759796)

[3.6.2 Quản lý các đối tượng khác ngoài phạm vi người chơi 56](#_Toc507759797)

[3.7 Xây dựng hành động bắn trực tuyến 59](#_Toc507759798)

[3.7.1 Thiết lập công cụ mạng cho mô hình đạn 60](#_Toc507759799)

[3.7.2 Viết lệnh Command cho client 61](#_Toc507759800)

[3.7.3 Viết lệnh ServerCallback của server 61](#_Toc507759801)

[3.8 Tổng kết chương 62](#_Toc507759802)

[CHƯƠNG 4 – XÂY DỰNG VR CHO TRÒ CHƠI 63](#_Toc507759803)

[4.1 Kiến nghị về phần cứng và phần mềm 63](#_Toc507759804)

[4.1.1 Kiến nghị về phần cứng 63](#_Toc507759805)

[4.1.2 Kiến nghị về phần mềm 64](#_Toc507759806)

[4.2 Thiết lập VR cho trò chơi 64](#_Toc507759807)

[4.3 Thiết lập VR cho nhân vật 66](#_Toc507759808)

[4.3.1 Thêm các đoạn script VR cơ bản 66](#_Toc507759809)

[4.3.2 Tạo camera VR cho người chơi 68](#_Toc507759810)

[4.3.3 Tạo độ dãn hình ảnh khi sử dụng VR 70](#_Toc507759811)

[4.4 Thiết lập và chạy ứng dụng 70](#_Toc507759812)

[4.4.1 Xuất bản (Build) trò chơi 70](#_Toc507759813)

[4.4.2 Chạy trò chơi trên nền mạng LAN 72](#_Toc507759814)

[4.5 Kết luận 72](#_Toc507759815)

DANH MỤC KÍ HIỆU VÀ CHỮ VIẾT TẮT

**CÁC KÝ HIỆU**

**CÁC CHỮ VIẾT TẮT**

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

**DANH MỤC HÌNH**

[Hình 1.1 Nguyên lý hoạt động của VR[1] 11](#_Toc507759816)

[Hình 2.1 Mục tiêu hoàn thành của người chơi 14](#_Toc507759817)

[Hình 2.2 Bối cảnh sơ lược của thành phố 15](#_Toc507759818)

[Hình 2.3 Vị trí ban đầu của người chơi và vị trị hoàn thành mục tiêu 15](#_Toc507759819)

[Hình 2.4 Mô hình mẫu nhân vật của người chơi 16](#_Toc507759820)

[Hình 2.5 Lượng máu ban đầu người chơi được cung cấp 16](#_Toc507759821)

[Hình 2.6 Các chỉ số tấn công của người chơi 17](#_Toc507759822)

[Hình 2.7 Các chỉ số khác của người chơi 17](#_Toc507759823)

[Hình 2.8 Phép kiểm tra độ dịch chuyển tọa độ Ox, Oz 18](#_Toc507759824)

[Hình 2.9 Thực hiện di chuyển nhân vật 18](#_Toc507759825)

[Hình 2.10 Lệnh kiểm tra có thực hiện nhảy hay không 19](#_Toc507759826)

[Hình 2.11 Lệnh nhảy cho nhân vật 19](#_Toc507759827)

[Hình 2.12 Kiểm tra xem người dùng có thực hiện bắn hay không 20](#_Toc507759828)

[Hình 2.13 Lệnh kiểm tra số lượng đạn hiện thời 21](#_Toc507759829)

[Hình 2.14 Đoạn lệnh thực hiện bắn đạn 21](#_Toc507759830)

[Hình 2.15 Hàm tạo khoảng cách thời gian bắn giữa 2 viên đạn 21](#_Toc507759831)

[Hình 2.16 Thực hiện nạp đạn 22](#_Toc507759832)

[Hình 2.17 Cấu trúc hàm WaitReload 23](#_Toc507759833)

[Hình 2.18 Mô hình ZombieAA 24](#_Toc507759834)

[Hình 2.19 Mô hình ZombieA 24](#_Toc507759835)

[Hình 2.20 Mô hình ZombieB 25](#_Toc507759836)

[Hình 2.21 Mô hình ZombieC 26](#_Toc507759837)

[Hình 2.22 Mô hình ZombieD 27](#_Toc507759838)

[Hình 2.23 Mô hình ZombieE\_V1 27](#_Toc507759839)

[Hình 2.24 Mô hình ZombieE\_V2 28](#_Toc507759840)

[Hình 2.25 Cấu trúc của 1 chuỗi automat hoạt động của Zombie 28](#_Toc507759841)

[Hình 2.26 Bật trạng thái Crawl của zombie 29](#_Toc507759842)

[Hình 2.27 Bật trạng thái KnockBack của zombie 29](#_Toc507759843)

[Hình 2.28 Thiết lập trạng thái Dead cho zombie 30](#_Toc507759844)

[Hình 2.29 Thiết lập thời gian chạy hiệu ứng Dead và xóa mô hình khi thực hiện xong 30](#_Toc507759845)

[Hình 2.30 Khai báo sử dụng thư viện AI của Unity 30](#_Toc507759846)

[Hình 2.31 Kiểm tra xem người chơi có nằm trong tầm ngắm của zombie không 31](#_Toc507759847)

[Hình 2.32 Zombie sẽ tấn công nếu người chơi nằm trong phạm vi rangeAttack 31](#_Toc507759848)

[Hình 2.33 Bật trạng thái di chuyển của zombie 31](#_Toc507759849)

[Hình 2.34 Cấu trúc hàm NearestPlayer 32](#_Toc507759850)

[Hình 2.35 Gán lại đối tượng mục tiêu của zombie 32](#_Toc507759851)

[Hình 2.36 Trừ máu của người chơi khi bị zombie tấn công 33](#_Toc507759852)

[Hình 3.1 Cấu tạo của HLAPI[3] 36](#_Toc507759853)

[Hình 3.2 Quy trình hoạt động của multiplayer networking[4] 38](#_Toc507759854)

[Hình 3.3 Thành phần công cụ NetworkManger của Unity 42](#_Toc507759855)

[Hình 3.4 Trình định danh Network Identity 42](#_Toc507759856)

[Hình 3.5 Đăng kí nhân vật của người chơi lên Network Controller 43](#_Toc507759857)

[Hình 3.6 Mối quan hệ giữa player, client và server[4] 45](#_Toc507759858)

[Hình 3.7 Công cụ Network Transform của Unity 46](#_Toc507759859)

[Hình 3.8 Mối quan hệ giữa Client, Player, Server và đối tượng không thuộc người chơi[4] 46](#_Toc507759860)

[Hình 3.9 Tạo một game object rỗng 48](#_Toc507759861)

[Hình 3.10 Thêm công cụ NetworkManager vào đối tượng 48](#_Toc507759862)

[Hình 3.11 Tạo mô hình thô cho nhân vật 49](#_Toc507759863)

[Hình 3.12 Thêm trình điều khiển hoạt động cho nhân vật 49](#_Toc507759864)

[Hình 3.13 Animator sau khi thiết lập 50](#_Toc507759865)

[Hình 3.14 Các đoạn script đã được kéo vào 50](#_Toc507759866)

[Hình 3.15 Box Collider sau khi được thêm vào nhân vật 50](#_Toc507759867)

[Hình 3.16 Sau khi thêm Audio Source vào nhân vật 50](#_Toc507759868)

[Hình 3.17 Thêm NetworkIdentity vào nhân vật 51](#_Toc507759869)

[Hình 3.18 Ủy quyền cho máy người chơi quản lý nhân vật 51](#_Toc507759870)

[Hình 3.19 Thêm trình đồng bộ hóa vào nhân vật 51](#_Toc507759871)

[Hình 3.20 Trình đồng bộ hóa của nhân vật 52](#_Toc507759872)

[Hình 3.21 Cách chuyển từ 1 script thường sang script mạng 52](#_Toc507759873)

[Hình 3.22 Đoạn code lọc bỏ các đối tượng không thuộc người chơi 52](#_Toc507759874)

[Hình 3.23 Tắt các camera của các người chơi khác 53](#_Toc507759875)

[Hình 3.24 Khai báo Camera và các thuộc tính liên quan 54](#_Toc507759876)

[Hình 3.25 Khai báo vị trí bắt đầu quay của camera 54](#_Toc507759877)

[Hình 3.26 Khai báo góc quay của của camera 54](#_Toc507759878)

[Hình 3.27 Bật cờ thông báo hủy camera 55](#_Toc507759879)

[Hình 3.28 Thông báo bật camera khi người chơi thoát khỏi trận đấu 55](#_Toc507759880)

[Hình 3.29 Tắt camera nếu cờ isRotate là false 55](#_Toc507759881)

[Hình 3.30 Khởi động camera và tạo góc xoay cho camera 56](#_Toc507759882)

[Hình 3.31 Xoay và dịch chuyển camera 56](#_Toc507759883)

[Hình 3.32 Định danh mạng cho đối tượng và ủy quyền cho server quản lý 57](#_Toc507759884)

[Hình 3.33 Khởi tạo các thuộc tính cho lớp tạo zombie 57](#_Toc507759885)

[Hình 3.34 Hàm thiết lập khu vực 58](#_Toc507759886)

[Hình 3.35 Hàm tạo zombie tự động khi trận đấu mạng bắt đầu 59](#_Toc507759887)

[Hình 3.36 Thêm thư viện UnityEngine.Networking cho lớp 59](#_Toc507759888)

[Hình 3.37 Chuyển sang kế thừa lớp NetworkBehaviour 59](#_Toc507759889)

[Hình 3.38 Thêm NetworkIdentity cho mô hình Bullet 60](#_Toc507759890)

[Hình 3.39 Thêm NetworkTransform 60](#_Toc507759891)

[Hình 3.40 Thêm Bullet vào Registered Spawnable Prefabs 61](#_Toc507759892)

[Hình 3.41 Cấu trúc hàm sinh đạn trên hệ thống mạng 61](#_Toc507759893)

[Hình 3.42 Cấu trúc hàm giải phóng đạn ra khỏi mạng và xóa mô hình đạn 62](#_Toc507759894)

[Hình 4.1 Thiết lập cho nhân vật hỗ trợ VR 65](#_Toc507759895)

[Hình 4.2 Bấm chọn chế độ hỗ trợ VR cho nhân vật 65](#_Toc507759896)

[Hình 4.3 Tạo một game object rỗng 66](#_Toc507759897)

[Hình 4.4 Thêm VR vừa tải vô game object của player 67](#_Toc507759898)

[Hình 4.5 Thêm script GvrHead vào Head 68](#_Toc507759899)

[Hình 4.6 Thêm thành phần Camera cho Main Camera 69](#_Toc507759900)

[Hình 4.7 Thêm Gvr Eye cho Main Camera Left 69](#_Toc507759901)

[Hình 4.8 Thêm Gvr Eye cho Main Camera Right 69](#_Toc507759902)

[Hình 4.9 Thêm Gvr Reticle Pointer cho GvrReticalPointer 70](#_Toc507759903)

[Hình 4.10 Thêm Gvr Pointer Manager và Gvr Pointer Input Module vào pointerwalid. 70](#_Toc507759904)

[Hình 4.11 Mở chức năng Build của Unity 71](#_Toc507759905)

[Hình 4.12 Chọn thiết bị mà trò chơi sẽ chạy trên nó 71](#_Toc507759906)

CHƯƠNG 1 – TỔNG QUAN VỀ VIRTUAL REALITY

1.1 Khái niệm về Virtual Reality

Thực tế ảo (Virtual Reality - VR) là một công nghệ máy tính sử dụng các thiết bị thực tế ảo hoặc các thiết bị đa chiều, đôi khi kết hợp với các thiết bị vật lý hoặc đạo cụ, để tạo ra hình ảnh, âm thanh và các cảm giác thực tế cho người dùng từ những hiện tượng, sự vật hay hình ảnh được mô phỏng một cách vật lý từ các môi trường ảo hoặc tưởng tượng (trò chơi điện tử (game), hình ảnh 360O,…)[1].

Một người sử dụng thiết bị thực tế ảo có thể nhìn thấy toàn bộ thế giới nhân tạo (mô phỏng), và với thiết bị VR chất lượng cao còn có thể cho phép chúng ta di chuyển và tương tác với môi trường (thế giới) ảo đó. Các hiệu ứng VR này sẽ được tạo ra bởi thiết bị đeo VR có cấu trúc gồm 1 màn hình đặt ngay trước mắt (sát mắt), nhưng cũng có thể được tạo ra thông qua các không gian được thiết kế đặc biệt được hình thành từ nhiều màn hình lớn gộp lại.

Hệ thống VR bao gồm bộ truyền tải rung và các cảm biến khác cho người dùng thông qua các bộ điều khiển trò chơi hoặc các thiết bị khác được gọi là các hệ thống haptic. Các thông tin mà ta cảm nhận được này được gọi là các lực hồi tiếp (force feedback) trong các ứng dụng về y tế, trò chơi video và huấn luyện quân sự. VR cung cấp các hiện diện ảo cho người dùng thông qua các thiết bị tay cầm và nghe nhìn để khiến người dùng được cảm giác như thật với môi trường được mô phỏng. Người dùng sẽ được đắm chìm vào môi trường ảo nhưng lại khiến người dùng cảm thấy rằng mình đang ở thế giới thực để tạo ra các giá trị trải nghiệm thực tiễn cho người dùng làm thỏa mãn niềm khát khao về khoa học viễn tưởng, huyền thoại, cổ tích, hay chỉ là để đáp ứng các nhu cầu giải trí thông thường nói chung mục đích của VR chính là đáp ứng được mong muốn từ lâu của nhiều người đó là họ có thể đắm mình vào thế giới ảo và họ có thể hòa mình vào nó. Môi trường VR sẽ gần giống với môi trường thực tế dựa trên các hiệu ứng về vật lý, sinh học, địa lý, mỹ thuật,… Vì vậy tính chất quan trọng nhất của VR đó là phải làm sao khiến con người có thể hòa nhập vào được vào thế giới ảo bằng các giác quan (với công nghệ hiện tại tạm thời chúng ta chỉ có thể nghe được, nhìn được thế giới ảo). Các hệ thống thực tế tăng cường cũng có thể được coi là một hình thức VR mà lớp thông tin ảo (yếu tố ảo) sẽ được lồng vào các khung cảnh thật hay clip, video quay cảnh thật được ghi lại bằng máy ảnh có hỗ trợ tai nghe hoặc qua điện thoại thông minh hoặc máy tính bảng.

1.2 Nguồn gốc của thuật ngữ

Năm 1938, trong tập các bài luận Le Théâtre et son double, Antonin Artaud trình bày các yếu tố ảo tự nhiên của các nhân vật và các đối tượng trong nhà hát và ông gọi các yếu tố là "la réalité virtuelle" mà ngày nay chúng ta gọi là VR. Bản dịch tiếng Anh của cuốn sách này, được xuất bản vào năm 1958 đã chuyển thể cụm từ "la réalité virtuelle" thành "virtual reality". Thuật ngữ "thực tế nhân tạo" (artificial reality), do Myron Krueger đặt ra, đã được sử dụng từ những năm 1970. Và cuối cùng, thuật ngữ "thực tế ảo" (VR) đã được sử dụng bắt đầu từ cuốn tiểu thuyết khoa học The Judas Mandala vào năm 1982 của Damien Broderick.[1]

1.3 Công nghệ

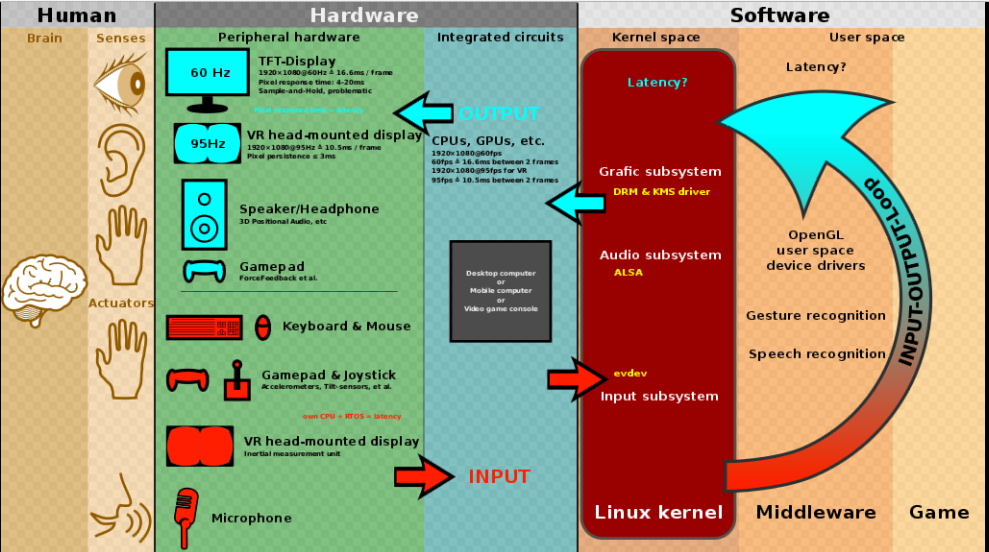
Ngôn ngữ mô hình thực tế ảo (Virtual Reality Modeling Language - VRML) được giới thiệu lần đầu tiên vào năm 1994 nhằm mục đích xây dựng nên "thế giới ảo" mà người dùng có thể hòa mình vào không gian đó mà không cần phụ thuộc vào tai nghe. Tổ chức Web3D sau đó được thành lập năm 1997 để phát triển các tiêu chuẩn công nghiệp dành cho đồ hoạ 3D chạy trên web. Tổ chức sau đó đã tạo ra X3D từ framework VRML như là một tiêu chuẩn lưu trữ và cũng là tiêu chuẩn mã nguồn mở cho việc phân phối nội dung VR trên các trang web.

Tất cả các công nghệ hiện nay đều tập trung phát triển cho các VR chạy trên nền tảng là điện thoại thông minh, các công nghệ này bao gồm: kính ngắm (kính VR) và các cảm biến chuyển động để theo dõi như đầu, tay, và các cảm biến khác ở các vị trí khác của cơ thể; các kính VR này có màn hình là màn hình HD và 2 mắt của nó là các thấu kính lồi (áp dụng phương pháp lập thể - stereoscopy); ngoài ra kính VR được trang bị thêm một bộ vi xử lý nhỏ, nhẹ và nhanh. Những thành phần này đã tạo nên thiết bị VR Oculus Rift 2012 đầu tiên trên thế giới.

Việc sản xuất hình ảnh và video VR độc lập đã tăng lên nhờ sự phát triển của các máy quay đa hướng, hay còn được gọi là các máy ảnh 360 độ hoặc máy quay VR, có khả năng ghi lại tất cả các hướng của một cảnh ngoài đời thực. Với công nghệ hiện tại cho dù ở độ phân giải thấp hay ở định dạng nén cao để truyền trực tuyến ta đều có thể tạo ra các hình ảnh VR này. Thêm vào đó, kĩ thuật chụp ảnh tĩnh bằng quang học cổ điển sẽ được sử dụng bằng cách chụp nhiều bức ảnh có độ phân giải cao và kết hợp chúng lại để tạo ra các vật thể và môi trường 3D chi tiết trong các ứng dụng VR.

1.4 Nguyên lý hoạt động của VR

Ta có bức ảnh sau:



Hình 1.1 Nguyên lý hoạt động của VR[1]

Nguyên lý hoạt động của VR gồm 3 phần chính là con người (human), các thiết bị phần cứng (hardware) và các phần mềm (software).

Một VR đầy đủ sẽ có 2 giai đoạn. Giai đoạn 1 bắt đầu từ việc con người sẽ sinh ra dữ liệu tương tác và các dữ liệu sẽ được truyền đi cho đến điểm kết thúc ở giai đoạn 1 là chương trình ứng dụng (game, ứng dụng phim, ảnh,…). Và giai đoạn 2 bắt đầu từ việc chương trình ứng dụng sẽ sinh ra các hình ảnh, phim, hoạt động mang tính VR và điểm kết thúc sẽ là sự cảm nhận, nhận thấy, giác ngộ các mô hình giả lập này ở con người.

Giai đoạn 1 này sẽ như sau:

* Đầu tiên, con người sẽ tương tác với chương trình thông qua giọng nói, cử chỉ hoặc các tương tác tay (da).
* Sau đó các dữ liệu tương tác này của con người sẽ được các thiết bị đầu vào (chuột, bàn phím, micro, các thiết bị cảm ứng, kính VR,…) thu thập lại.
* Tiếp đó, các dữ liệu này sẽ được chuyển đến các mạch như CPU, GPU, và các mạch tích hợp khác để xử lý các kỹ thuật vật lý.
* Tiếp theo các dữ liệu này sẽ được chuyển đến kernel để tạo các cơ chế liên lạc giữa các tiến trình (chương trình VR đang chạy với các thiết bị phần cứng như CPU, GPU,… Ở đây, trong một vài trường hợp sẽ xuất hiện độ trễ tùy vào dữ liệu hay hệ thống.
* Sau khi tạo xong các cơ chế liên lạc, dữ liệu sẽ được chuyển đến các chương trình nhận dạng (nhận dạng cử chỉ, giọng nói,…), xử lý và phân tích các dữ liệu tương tác này. Tùy vào độ phức tạp mà chương trình sẽ xuất hiện độ trễ hay là không.
* Cuối cùng dữ liệu sẽ được truyền đến điểm cuối cùng là các chương trình ứng dụng.

Giai đoạn 2 VR sẽ hoạt động như sau:

* Nếu có dữ liệu đầu vào, thì các chương trình ứng dụng sẽ phân tích và xử lý và trả ra kết quả tương ứng. Còn nếu không xảy ra giai đoạn 1 thì chương trình sẽ sinh ra các dữ liệu mang tính VR (hình ảnh VR (360 độ), phim VR,…).
* Sau đó dữ liệu sẽ được chuyển đến kernel để tạo cầu nối kết nối tới các thiết bị phần cứng.
* Sau khi được kernel chuyển tiếp thì tại đây các thiết bị phần cứng như CPU, GPU,… sẽ tạo ra hình ảnh, âm thanh,…từ các dữ liệu này.
* Dữ liệu sau khi được mô phỏng hóa sẽ được trình bày, biểu đạt hay hiện thị trên các thiết bị đầu ra như kính VR, màn hình, loa,…
* Và cuối cùng sẽ tiếp nhận, xử lý và cảm nhận các hình ảnh, phim, âm thanh đã được mô phỏng hóa này.

Một VR không nhất cần phải có cả 2 giai đoạn, tuy nhiên giai đoạn 2 là bắt buộc vì mục tiêu cơ bản nhất và quan trọng nhất là biểu hiện, trình bày và mô phỏng hóa để con người có thể “hòa mình” vào phong cảnh ảo hóa mà chương trình đã tạo ra.

CHƯƠNG 2 – TỔNG QUAN VỀ TRÒ CHƠI

2.1 Các thông tin chung về trò chơi

2.1.1 Thông tin về trò chơi

Tên của trò chơi: The Lost Land (Vùng đất chết).

Thể loại của trò chơi: First-person Action (Hành động góc nhìn thứ nhất).

Nền tảng: Android với Oculus VR, Google CardBoard hoặc Samsung Gear VR.

2.1.2 Bối cảnh của trò chơi

The Lost Land là một trò chơi hành động góc nhìn thứ nhất. Người chơi sẽ đóng vào vai nhân vật là một người lính Mỹ bị bỏ hoang ở một vùng đất chết. Vùng đất chết này là nhà của những kẻ thù nguy hiểm là những con zombie chết người.

2.1.3 Mục tiêu hoàn thành của trò chơi

Người chơi sẽ cố gắng làm sao đưa nhân vật của mình trở về nhà một cách an toàn. Để làm được điều này, người chơi sẽ phải điều khiển nhân vật chạy tới chỗ trực thăng cứu hộ mà người lính này đã gọi cứu viện.

Hoặc ta cũng có thể thắng bằng cách tiêu diệt hết Zombie ở trên bản đồ.



Hình 2.1 Mục tiêu hoàn thành của người chơi

2.1.4 Các tương tác của người dùng

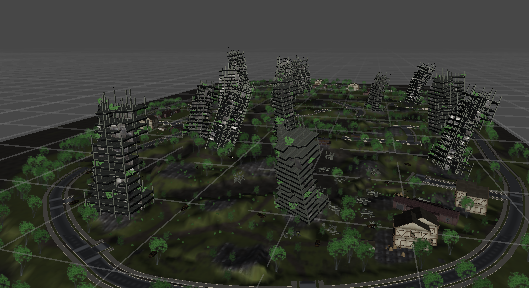
Người dùng sẽ tương tác với trò chơi bằng cách sử dụng thiết bị VR đeo lên đầu để xem được khung nhìn mà nhân vật đang thấy. Để có thể di chuyển, nhảy hay là bắn, người chơi sẽ sử dụng bộ điều khiển cầm tay có hỗ trợ VR.

Ngoài ra trò chơi có hỗ trợ chế độ chơi 2 người trên mạng LAN. 2 người chơi với nhau trên cùng một bản đồ thông qua kết nối chung trên cùng một mạng.

2.2 Thông tin về bản đồ, nhân vật và kẻ thù

2.2.1 Thông tin chung về bản đồ

Bản đồ của trò chơi là một thành phố đã bị tàn phá, và bỏ hoang.



Hình 2.2 Bối cảnh sơ lược của thành phố



Hình 2.3 Vị trí ban đầu của người chơi và vị trị hoàn thành mục tiêu

Ở trên hình 2.3, dấu hoa thị đỏ tượng trưng cho vị trí ban đầu của người chơi và vòng tròn đỏ là nơi mà trực thăng cứu hộ đang chờ sẵn. Người chơi sẽ phải đi từ chỗ có dấu hoa thị (\*) đến vị trí dấu tròn (O) để gặp trực thăng cứu hộ để hoàn thành mục tiêu (thắng) trò chơi.

2.2.2 Thông tin chung về nhân vật

Nhân vật mà người chơi điều khiển là một người lính Mỹ. Người lính có thể đi, nhảy, bắn sao cho có thể đến được vị trí đích.

Nhân vật của người chơi sẽ có một thuộc tính có tên là blood (hay hp) có nghĩa máu. Ban đầu người chơi sẽ được cung cấp một lượng máu cụ thể (là một con số lớn hơn 0). Cứ mỗi lần bị kẻ thù tấn công, lượng máu (chỉ số máu) của người chơi sẽ bị giảm. Nếu lượng máu trở về 0, người chơi sẽ thua cuộc.

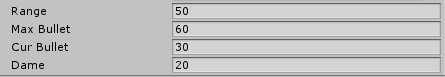


Hình 2.4 Mô hình mẫu nhân vật của người chơi



Hình 2.5 Lượng máu ban đầu người chơi được cung cấp

Nhân vật người chơi còn có các chỉ số tấn công (Dame), tầm bắn (Range), số lượng đạn hiện thời (Cur Bullet) và số lượng mỗi lần nạp đạn (số lượng đạn tối đa) (Max Bullet).



Hình 2.6 Các chỉ số tấn công của người chơi

Ngoài ra người chơi còn có các chỉ số như tốc độ di chuyển (speed), trọng lượng (gravity) và tốc độ nhảy (jump speed).



Hình 2.7 Các chỉ số khác của người chơi

2.2.3 Thông tin chung về kẻ thù

Kẻ thù là những nhân vật sẽ tấn công người chơi, làm giảm đi lượng máu mà người chơi đang có. Kẻ thù của người chơi là những con zombie. Có tất cả là 7 loại zombie với các mô hình khác nhau là: ZombieAA, ZombieA, ZombieB, ZombieC, ZombieD, ZombieE\_V1 và Zombie E\_V2.

Mỗi zombie sẽ có các thuộc tính:

* zombieName: tên của zombie
* health: máu của zombie (máu của zombie sẽ bị giảm bằng đúng lượng (chỉ số) Dame của người chơi khi người chơi bắn trúng vào zombie)
* rangeFollow: phạm vi phát hiện người chơi
* rangeAttack: phạm vi (khoảng cách) mà zombie tấn công người chơi

2.3 Chi tiết về nhân vật

2.3.1 Xây dựng di chuyển cho nhân vật

Người chơi sẽ di chuyển nhân vật bằng cách sử dụng tay cầm điều khiển VR.

Về phần chương trình, Ta sẽ khiến nhân vật có thể di chuyển bằng cách dùng lệnh Transform.Translate(x, y, z) để di chuyển mô hình của người chơi. Trong đó:

* x: là dịch chuyển mô hình theo trục Ox
* y: là dịch chuyển mô hình theo trục Oy
* z: là dịch chuyển mô hình theo trục Oz.

Để biết khi nào cần dùng phép translate, ta sẽ dùng các bộ nhận biết di chuyển dựa vào các thao tác trên các thiết bị đầu vào của người dùng. Unity có hỗ trợ ta vấn đề này bằng hàm Input. GetAxis(string axisName) với axisName là trục tọa độ mà người dùng muốn di chuyển nhân vật.

Do ở đây là phép xây dựng di chuyển trên mặt đất, vì vậy ta sẽ xét xem độ dịch chuyển theo trục hoành (Ox) và trục tung (Oz) có khác 0 hay không. Ta sẽ thực hiện bằng câu lệnh sau:



Hình 2.8 Phép kiểm tra độ dịch chuyển tọa độ Ox, Oz

Nếu dữ liệu nhập đầu vào của người dùng làm dịch chuyển nhân vật theo chiều của trục Ox hoặc Oz (độ dịch chuyển Input.GetAxis() khác 0) thì ta sẽ thực hiện phép dịch chuyển translate như sau:



Hình 2.9 Thực hiện di chuyển nhân vật

Do đây là di chuyển trên mặt đất nên ta sẽ chỉ di chuyển nhân vật theo chiều của trục hoành (Ox) và trục tung (Oz) vì vậy trục cao (Oy) sẽ có giá trị là 0. Độ lớn của phép dịch trên Ox và Oz sẽ là độ dịch chuyển đầu vào từ các thiết bị của người nhập (vận tốc nhập của thiết bị) nhân với Time.deltaTime (thời gian chuyển cảnh giữa 2 frame) với chỉ số speed của nhân vật (chỉ số bội). Chỉ số bội càng cao càng giúp nhân vật di chuyển được khoảng cách xa hơn. Nói chung phép translate phù hợp và logic với công thức vật lý.

2.3.2 Xây dựng hoạt động nhảy của nhân vật

Tương tự với thao tác nhảy, tuy nhiên nhảy là thao tác nhân vật đưa thân mình lên trên mặt đất, tức là chỉ dịch chuyển theo theo trục cao (Oy). Vì vậy ta sẽ xét xem người chơi có làm thay đổi độ dịch chuyển đầu vào từ các thiết bị nhập theo chiều của trục Oy hay không. Vì ta cũng cần xét thêm người chơi có đang đứng trên mặt đất hay không. Ta có lệnh kiểm tra như sau:



Hình 2.10 Lệnh kiểm tra có thực hiện nhảy hay không

Nếu thỏa mãn điều kiện, ta sẽ thực hiện động tác nhảy cho nhân vật bằng cách thay đổi thuộc tính velocity của bộ cấu trúc vật lý RigidBody của nhân vật để dịch chuyển nhân vật theo chiều của trục Oy.



Hình 2.11 Lệnh nhảy cho nhân vật

jumpSpeed là chỉ số tốc độ nhảy của nhân vật.

Ban đầu nhân vật đang ở dưới đất. Ta gán vector velocity với vector dịch chuyển là Vector3(0f, jumpSpeed, 0f) tức là ta tạo ra một thao tác hành động có vector vận tốc là Vector3(0f, jumpSpeed, 0f). Do nhân vật di chuyển thẳng đứng mà ta lại muốn dịch chuyển theo phương thẳng đứng nên hệ số góc của nó là 0 mà cos(0o) = 1 vì vậy độ lớn của vận tốc mà ta muốn tính sẽ bằng chính độ lớn của vector vận tốc mà ta đã gán. Dễ hiểu hơn đó là nhân vật sẽ di chuyển theo hướng của vector vận tốc và di chuyển với độ lớn vận tốc bằng chính độ lớn của vector.

2.3.3 Tạo khung nhìn cho nhân vật

Camera là một công cụ giúp người chơi có thể thấy những gì xảy ra trong trò chơi. Nó giống như máy quay phim ngoài đời thực. Bản chất mắt của con người chính là một máy quay phim do tự nhiên tạo ra. Vì thế ta sẽ tạo “mắt” cho nhân vật người chơi bằng cách gắn Camera lên người chơi, khi đó nhân vật di chuyển đến đâu thì camera đi theo đó, giúp người chơi có thể thấy được những gì mà nhân vật người chơi thấy.

2.3.4 Xây dựng hoạt động bắn cho nhân vật

Bước tiếp theo, ta sẽ tiến hành đi xây dựng thao tác bắn cho nhân vật.

Đầu tiên, ta sẽ xét xem người dùng có bấm nút bằng không bằng cách dùng hàm kiểm tra dữ liệu đầu vào của Unity.



Hình 2.12 Kiểm tra xem người dùng có thực hiện bắn hay không

Nếu thỏa mãn điều kiện ta sẽ tiến hành lập trình thao tác bắn đạn.

Trước hết ta cần phải tìm hiểu khái niệm mới đó là Coroutine.

Khi ta thực thi một hàm, hàm đó sẽ được xử lý hoàn toàn trước khi trả về một giá trị. Điều đó có nghĩa là dù hàm có phức tạp đến mấy cũng sẽ được xử lý trong duy nhất một vòng lặp của chương trình. Do đó, các hàm dạng này không thể được sử dụng để xử lý các hiệu ứng, animation hoặc các sự kiện diễn ra trong nhiều thời gian.

Trong chương trình máy tính, entry point là một điểm mà tại đó bộ vi xử lý nhập vào một chương trình hoặc một đoạn mã và bắt đầu thực thi đoạn mã hay chương trình đó.

Coroutine là một thành phần của chương trình máy tính giúp chúng ta tạo ra các tiến trình con độc lập, bằng cách cho phép tại một vị trí có thể có nhiều entry point hoạt động trong cùng thời điểm. Trong Unity, Coroutine được sử dụng để bắt đầu một tiến trình độc lập với hàm Update và được gọi một lần duy nhất. Unity sẽ tự động quản lý tiến trình này cho đến khi nó kết thúc và được giải phóng.

Ta có thể hình dung việc bắn súng liên tục cũng vậy. Khi ta bắn súng liên hoàn đạn sẽ bay ra theo một chuỗi không ngừng. Vì vậy ta sẽ áp dụng tính Coroutine để tạo ra một chuỗi hình ảnh đạn bay.

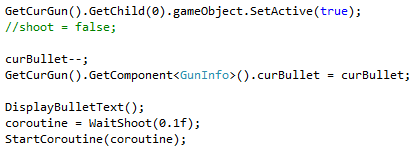
Nhưng trước khi bắn, ta phải kiểm tra súng còn đạn hay không bằng cách viết lệnh để kiểm tra số lượng đạn.



Hình 2.13 Lệnh kiểm tra số lượng đạn hiện thời

Nếu hết đạn ta sẽ thực hiện việc thay đạn (sẽ nói rõ ở phần 2.3.5).

Nếu không thỏa, ta sẽ bắt đầu việc bắn đạn.



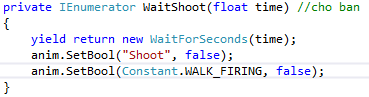
Hình 2.14 Đoạn lệnh thực hiện bắn đạn

Đầu tiên, ta sẽ bật nòng lên bằng cách gọi lệnh SetActive(true). Cứ mỗi lần bắn sẽ phải tốn 1 viên đạn vì vậy ta sẽ trừ số lượng đạn đi 1 bằng lệnh curBullet--;

Tiếp đến ta sẽ hiện thị số lượng đạn hiện hành lên UI cho người chơi biết bằng hàm DisplayBulletText();

Tiếp tục ta sẽ tạo chuỗi coroutine bằng cách gọi hàm WaitShoot(0.1f);

Hàm WaitShoot(0.1f) như sau:



Hình 2.15 Hàm tạo khoảng cách thời gian bắn giữa 2 viên đạn

Ta gặp ngay từ khóa khái niệm mới đó là yield. Vậy yield là gì?

Từ khoá yield được ứng dụng nhiều trong các ngôn ngữ lập trình. yield là một tín hiệu để báo hiệu cho trình biên dịch biết đoạn code hay hàm chứa nó chính là một khối lặp, mặc dù không sử dụng các cú pháp thông thường đã biết. yield sẽ kết hợp với return, cho phép trả về các giá trị của khối lặp đó, đồng thời có thể quay trở lại khối lặp trong những lần lặp tiếp theo.

yield return sẽ không kết thúc phương thức chứa nó mà vẫn tiếp tục chạy cho đến khi thực thi xong lệnh cuối cùng của khối lặp. Muốn kết thúc phương thức, ta sử dụng yield break. Phương thức chứa yield có kiểu trả về là IEnumerator, được sử dụng nhằm tạo ra một Coroutine.

Lệnh yield này sẽ lặp hàm WaitForSeconds() để khiến chương trình chờ một khoảng thời gian, khi hết đúng bằng thời gian được truyền vào, hàm này sẽ kết thúc và cho phép chương trình chạy tiếp. Giá trị time ta truyền vào tức là ta muốn thời gian chờ giữa 2 viên đạn được bắn ra trong chuỗi đạn hoặc giữa hành động bắn đạn với hành động khác có khoảng cách là 0.1 s. Kiểu trả về của hàm WaitShoot(time) là IEnumerator là kiểu cho phép hỗ trợ tạo ra một chuỗi hành động.

Như vậy sau khi thực hiện hàm WaitShoot(time) ta đã có thể tạo được một chuỗi đạn mà trong đó 2 viên đạn được bắn ra có thời gian cách nhau là 0.1s (nếu ta bắn liên hoàn).

Cuối cùng, ta gọi hàm StartCoroutine(coroutine) để chạy chuỗi đạn ta vừa mới tạo ra.

2.3.5 Xây dựng hoạt động nạp đạn cho nhân vật

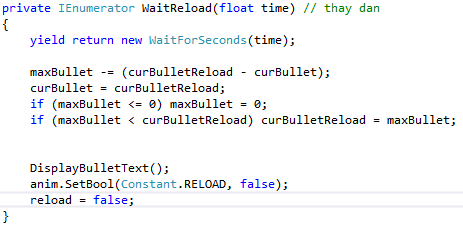
Ta thực hiện, việc nạp đạn bằng 2 dòng lệnh sau:



Hình 2.16 Thực hiện nạp đạn

Ở đây ta gọi hàm WaitReload(3.0f) để tạo hành động nạp đạn

Cấu trúc của hàm WaitReload(time) như sau:



Hình 2.17 Cấu trúc hàm WaitReload

Ta cũng tương tự tạo thời gian chờ thao tác nạp đạn của nhân vật là 3.0 s.

Sau đó ta thực hiện hành động nạp đạn bằng lệnh StartCoroutine(coroutine).

2.4 Chi tiết về kẻ thù

2.4.1 Cụ thể từng loại zombie

Trò chơi có tất cả là 7 loại zombie.

Đầu tiên là ZombieAA.

ZombieAA có các tham số như sau:

* health (máu): 200
* rangeFollow (khoảng cách phát hiện người chơi): 25
* rangeAttack (khoảng cách với tới và đánh người chơi): 2.8
* Damage (chỉ số tấn công): 2
* Mô hình của ZombieAA:



Hình 2.18 Mô hình ZombieAA

Zombie loại thứ 2 là ZombieA.

ZombieA có các tham số như sau:

* health (máu): 250
* rangeFollow (khoảng cách phát hiện người chơi): 25
* rangeAttack (khoảng cách với tới và đánh người chơi): 2.8
* Damage (chỉ số tấn công): 1
* Mô hình của ZombieA:

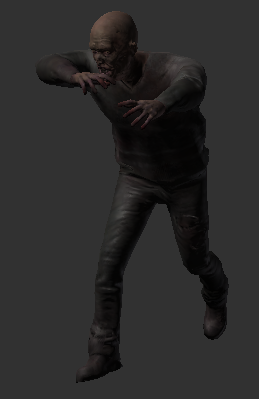


Hình 2.19 Mô hình ZombieA

Zombie loại thứ 2 là ZombieB.

ZombieB có các tham số như sau:

* health (máu): 180
* rangeFollow (khoảng cách phát hiện người chơi): 25
* rangeAttack (khoảng cách với tới và đánh người chơi): 2.8
* Damage (chỉ số tấn công): 1.5
* Mô hình của ZombieB:



Hình 2.20 Mô hình ZombieB

Zombie loại thứ 2 là ZombieC.

ZombieC có các tham số như sau:

* health (máu): 300
* rangeFollow (khoảng cách phát hiện người chơi): 30
* rangeAttack (khoảng cách với tới và đánh người chơi): 2.8
* Damage (chỉ số tấn công): 2
* Mô hình của ZombieC:

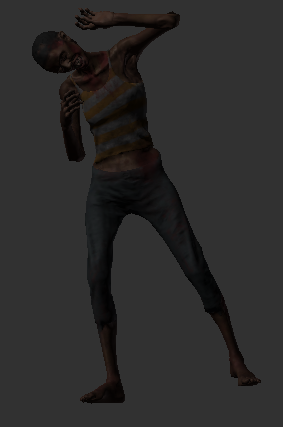


Hình 2.21 Mô hình ZombieC

Zombie loại thứ 2 là ZombieD.

ZombieD có các tham số như sau:

* health (máu): 265
* rangeFollow (khoảng cách phát hiện người chơi): 30
* rangeAttack (khoảng cách với tới và đánh người chơi): 2.8
* Damage (chỉ số tấn công): 1
* Mô hình của ZombieD:



Hình 2.22 Mô hình ZombieD

Zombie loại thứ 2 là ZombieE\_V1.

ZombieE\_V1 có các tham số như sau:

* health (máu): 500
* rangeFollow (khoảng cách phát hiện người chơi): 20
* rangeAttack (khoảng cách với tới và đánh người chơi): 2.8
* Damage (chỉ số tấn công): 2
* Mô hình của ZombieE\_V1:



Hình 2.23 Mô hình ZombieE\_V1

Zombie loại thứ 2 là ZombieE\_V2.

ZombieE\_V2 có các tham số như sau:

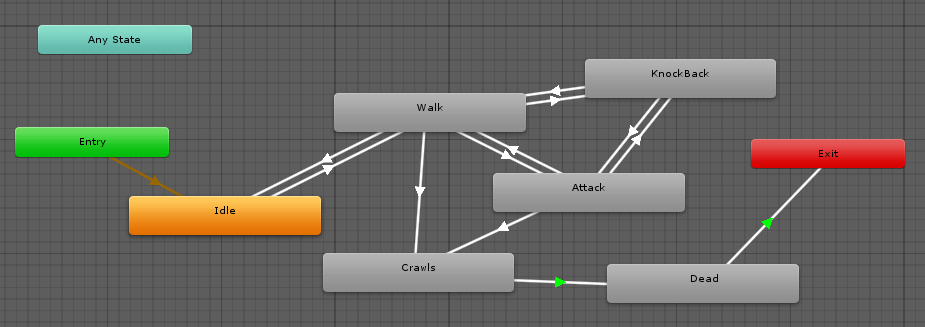
* health (máu): 1000
* rangeFollow (khoảng cách phát hiện người chơi): 15
* rangeAttack (khoảng cách với tới và đánh người chơi): 2.8
* Damage (chỉ số tấn công): 1.5
* Mô hình của ZombieE\_V2:



Hình 2.24 Mô hình ZombieE\_V2

2.4.2 Xây dựng chuỗi hoạt động cho kẻ thù

Trình quản lý hoạt động (Animator) của Zombie có dạng như sau:



Hình 2.25 Cấu trúc của 1 chuỗi automat hoạt động của Zombie

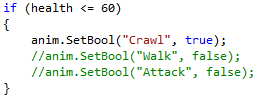
Dựa vào automat ta có thể zombie sẽ có các hoạt động cơ bản sau:

* Đứng yên (Idle)
* Di chuyển (Walk)
* Bò đất (Crawl)
* Dội ngược lại (KnockBack)
* Tấn công (Attack)
* Chết (Dead)

Trong đó Walk sẽ được trình bày ở mục 2.4.3 và Attack sẽ được trình bày ở mục 2.4.4.

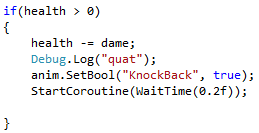
Đầu tiên, Idle là trạng thái bắt đầu của zombie nên ta không cần thiết lập bằng lệnh.

Crawl là trạng thái bò đất của zombie khi máu của zombie chỉ từ 60 trở xuống. Ta sẽ kích hoạt cho zombie bò bằng việc xét xem máu của zombie có nhỏ hơn hoặc bằng 60 hay không. Nếu thỏa mãn ta sẽ chuyển trạng thái hoạt động sang Crawl.



Hình 2.26 Bật trạng thái Crawl của zombie

KnockBack là trạng thái zombie thân sẽ bị dội ngược lại khi bị trúng đạn bởi người chơi. Nếu bị dính đạn mà máu giảm xuống 0 thì ta sẽ chuyển sang trạng thái Dead.



Hình 2.27 Bật trạng thái KnockBack của zombie

Trạng thái KnockBack có thời gian chạy hiệu ứng là 0.2 s.

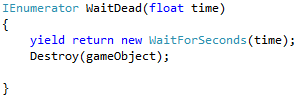


Hình 2.28 Thiết lập trạng thái Dead cho zombie

Vì khi chết zombie sẽ bị xóa ra khỏi bản đồ nên để tránh việc tranh chấp giữa các trạng thái khác, ta sẽ tắt tất cả các trạng khác trừ trạng thái Dead (false là tắt trạng thái, true là bật trạng thái).

Thời gian chạy hiệu ứng Dead là 2 s.

Sau khi hiệu ứng thực hiện xong, ta sẽ xóa mô hình bằng cách gọi hàm Destroy(gameObject);



Hình 2.29 Thiết lập thời gian chạy hiệu ứng Dead và xóa mô hình khi thực hiện xong

2.4.3 Xây dựng cách kẻ thù tìm người chơi

Đầu tiên, để sử dụng AI (trí tuệ nhân tạo) cho kẻ thù, ta khai báo thư viện:



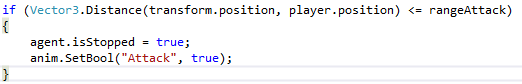
Hình 2.30 Khai báo sử dụng thư viện AI của Unity

Tính khoảng cách giữa người chơi với zombie. Nếu khoảng cách giữa zombie với người chơi nằm trong phạm vi phát hiện của zombie, thì zombie sẽ di chuyển về phía người chơi.



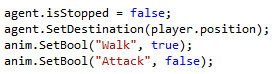
Hình 2.31 Kiểm tra xem người chơi có nằm trong tầm ngắm của zombie không

Trước khi di chuyển ta phải ưu tiên xem khoảng cách của người chơi với zombie có nằm trong phạm vi tấn công (rangeAttack) của zombie hay không. Nếu có, zombie sẽ tấn công người chơi.



Hình 2.32 Zombie sẽ tấn công nếu người chơi nằm trong phạm vi rangeAttack

Còn nếu nằm ngoài phạm vi tấn công của zombie, nhưng người chơi vẫn nằm trong vùng phát hiện của zombie thì ta sẽ chuyển trạng thái cho zombie sang Walk (di chuyển) đuổi theo người chơi.

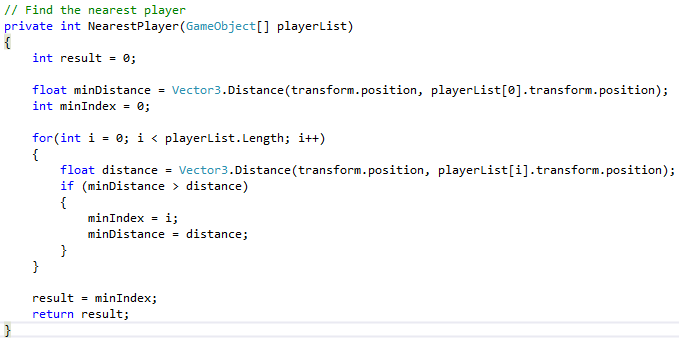


Hình 2.33 Bật trạng thái di chuyển của zombie

Trong đó, SetDestination(position) là hàm của Unity giúp đối tượng tìm đường đi tối ưu tới vị trí position. Hàm này trực thuộc thư viện UnityEngine.AI quản lý.

Sau khi tìm được đường đi ta sẽ bật chế độ di chuyển cho zombie bằng lệnh anim.SetBool(“Walk”,true);

Trong trường hợp mạng LAN sẽ có từ 2 người chơi trở lên. Nên ta phải giải quyết thêm vấn đề đó là zombie chỉ đuổi theo đối tượng nào có khoảng cách gần với zombie nhất. Ta thực hiện điều này bằng cách gọi hàm NearestPlayer(GameObject[] playerList)



Hình 2.34 Cấu trúc hàm NearestPlayer

Đầu tiên ta sẽ truyền danh sách chứa các nhân vật của tất cả người chơi.

Ta sẽ giả sử người chơi gần nhất với zombie là người chơi nằm ở vị trí đầu tiên trong mảng. Ta sẽ cho vòng lặp chạy từ 0 đến số lượng người chơi – 1. Nếu người chơi nằm ở vị trí i có khoảng cách với zombie nhỏ hơn khoảng cách nhỏ nhất thì ta sẽ gán lại khoảng cách nhỏ nhất bằng với khoảng cách giữa người chơi i với zombie.

Kết thúc vòng lặp ta tìm được người có khoảng cách gần nhất với zombie. Trả về giá trị người chơi đó.



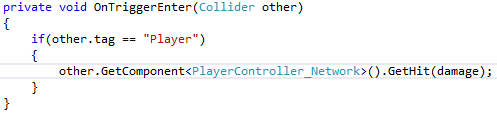
Hình 2.35 Gán lại đối tượng mục tiêu của zombie

Sau khi tìm được người chơi có khoảng cách gần nhất với zombie. Ta sẽ gán lại đối tượng mục tiêu của zombie. Ta sẽ gán đối tượng mục tiêu cho zombie đó là người chơi có khoảng cách gần với nó nhất mà ta đã tìm được.

2.4.4 Xây dựng cách kẻ thù tấn công người chơi

Ở chế độ tấn công người chơi sẽ tấn công người chơi. Lúc này mô hình zombie sẽ thực hiện các thao tác va chạm lên người chơi. Ta sẽ viết scrip xử lý va chạm. Mục đích của ta đó là lợi dụng va chạm này để gọi hàm trừ máu của người chơi.

Ta có đoạn lệnh như sau:



Hình 2.36 Trừ máu của người chơi khi bị zombie tấn công

Ta kiểm tra xem zombie có va chạm vào người chơi hay không. Nếu có tức là zombie đang tấn công lên người chơi. Cứ mỗi lần va chạm hay tấn công ta sẽ gọi hàm GetHit(damage) ở bên lớp PlayerController\_Network để trừ một lượng máu của người chơi bằng đúng giá trị damage.

2.5 Tổng kết chương

Kết thúc chương này ta có được những thông tin cơ bản về trò chơi như tên trò chơi, thể loại của trò chơi, mục tiêu cũng như bối cảnh của trò chơi.

Không những vậy ta còn biết được các thông tin về nhân vật người chơi như các chỉ số tấn công, di chuyển,… Bên cạnh đó, ta còn biết được cách xây dựng sao cho có thể di chuyển nhân vật, làm cho nhân vật nhảy, bắn hay thay đạn.

Ngoài ra, ta còn biết được các loại zombie sẽ xuất hiện trong trò chơi, các thông tin về chúng (chỉ số máu, tấn công,…) cũng như biết cách xây dựng các hoạt động cơ bản cho zombie như đi, tấn công, bò lăn, chết,…

CHƯƠNG 3 – XÂY DỰNG HỆ ĐA NGƯỜI CHƠI TRÊN NỀN MẠNG NỘI BỘ (LAN)

Multiplayer Networking (Mạng đa người chơi) là một trong những vấn đề phức tạp. Đa phần sự phức tạp này xuất phát từ các vấn đề về đồng bộ hóa và liên lạc giữa các trường hợp cùng xảy ra trên một bài toán (bối cảnh của trò chơi) mà mỗi trường hợp xảy ra xuất phát từ các đối tượng (người chơi) ở những máy tính khác nhau ở những nơi khác nhau trên trái đất.

Với các công cụ mạng đa người chơi (multiplayer networking) và công cụ High Level API (HLAPI) của Unity, ta có thể dễ dàng tạo ra một mạng lưới đa người chơi cho trò chơi của mình.

Chương này sẽ khái quát các khái niệm cũng như các nguyên tắc hoạt động cơ bản của các công cụ mà Unity sử dụng để xây dựng một mạng lưới đa người chơi. Đồng thời chương này sẽ hướng dẫn ta tạo ra một chế độ chơi đa người dùng trong một trò chơi cụ thể.

3.1 Tổng quan về mạng lưới đa người chơi của Unity

Có hai loại người dùng các tính năng Mạng lưới đa người dùng của Unity:

* Một là những nhà phát triển tạo trò chơi mạng lưới đa người dùng cơ bản. Những nhà phát triển này nên bắt đầu bằng việc sử dụng NetworkManager hoặc High Level API.
* Hai là những nhà phát triển có mục đích xây dựng cơ sở hạ tầng mạng hoặc các trò chơi có tính mạng lưới đa người dùng cao và phức tạp. Những nhà phát triển này nên bắt đầu bằng việc sử dụng nền tảng NetworkTransport API của Unity.

Do trò chơi ở đây chỉ xây dựng mạng lưới đa người chơi trên nền mạng nội bộ LAN nên tài liệu xin trình bày theo hướng thứ nhất ở trên, đó là dùng Network Manager hoặc High Level API.

3.1.1 High Level Scripting API (High Level API hay HLAPI)

Nền tảng mạng đa người dùng của Unity sử dụng các kịch bản (script) API "cấp cao" (**High Level Scripting API** hay còn gọi là **High Level API** (gọi tắt là **HLAPI**)). Nhà phát triển có thể sử dụng các kịch bản này để viết các dòng lệnh code để có thể xây dựng mạng lưới đa người chơi mà không cần phải lo lắng về cấu tạo chi tiết triển khai của các lớp thấp hơn lớp Transport trong mô hình mạng OSI. HLAPI cho phép ta:

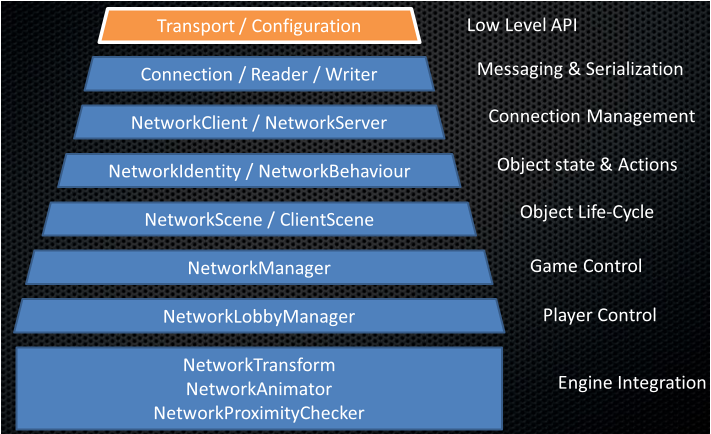
* Kiểm soát các trạng thái kết nối mạng của trò chơi bằng cách sử dụng công cụ "**Network Manager**" (Trình quản lý kết nối).
* Tạo ra một host cho mạng lưới mạng, mà máy chủ lưu trữ (host) này cũng chính là là một máy khách (một người chơi trong mạng lưới).
* Tuần tự dữ liệu bằng cách sử dụng trình tuần tự dữ liệu mục đích chung (general-purpose serializer).
* Gửi và nhận tin nhắn mạng.
* Gửi các lệnh kết nối mạng từ các máy khách (client) cho đến máy chủ (host).
* Thực hiện các cuộc gọi thủ tục từ xa từ máy chủ đến các máy khách.
* Gửi các sự kiện, thông điệp (event, command) mạng từ máy chủ đến các máy khách.

High Level API (HLAPI) là một hệ thống của Unity để hỗ trợ việc xây dựng mạng lưới đa người dùng (người chơi) cho các nhà sản xuất trò chơi. Nó được xây dựng ở phần đầu của nửa phần sau của lớp giao tiếp thời gian thực (transport), và xử lý được nhiều nhiệm vụ cơ bản mà một hệ thống đa người chơi đặt ra. Trong khi lớp vận chuyển (transport) có vai trò hỗ trợ cho các kết nối mạng (theo bất kì kiểu topo nào), thì HLAPI sẽ đóng vai trò là một hệ thống có thẩm quyền của máy chủ. Bằng việc cho phép một trong những người chơi tham gia (máy khách (client)) đóng vai trò là máy chủ host mà Unity không cần phải thiết lập quá trình máy chủ chuyên dụng. Ngoài ra nếu HLAPI kết hợp với các dịch vụ internet khác có thể giúp trò chơi được diễn ra một cách phong phú bởi nhiều người chơi thông qua internet mà không cần đến các nhà phát triển phải xây dựng, điều này vừa đảm bảo trò chơi vẫn có các dịch vụ internet mà vừa giúp các nhà phát triển tiết kiệm được thời gian và công sức.

Tóm gọn lại, HLAPI là một tập lệnh mạng mới được xây dựng gần đây trong Unity, hay nói cách khác nó chính là một trong những thư viện của Unity và tên của nó là UnityEngine.Networking. Mục tiêu của HLAPI đó là tập trung vào việc giúp nhà phát triển trò chơi dễ sử dụng và từ đó họ có thể phát triển các thể loại trò chơi một cách linh động và đồng thời HLAPI cũng cung cấp các dịch vụ hữu ích giúp các nhà sản xuất có thể dễ dàng tạo ra các trò chơi mạng lưới đa người chơi, các dịch vụ đó như:

* Trình xử lý tin nhắn
* Tuần tự hiệu suất cao các mục đích chung (General purpose high performance serialization)
* Quản lý phân tán các đối tượng
* Đồng bộ hóa các trạng thái
* Các lớp mạng bao gồm máy chủ (host), máy khách (client), và hỗ trợ các kết nối, vv

Cấu tạo của HLAPI được mô tả bằng lược đồ các chức năng sau:



Hình 3.1 Cấu tạo của HLAPI[3]

3.1.2 Trình soạn thảo và trình hoạt động tích hợp

Mạng được xây dựng trên nền tảng Unity được tích hợp với nó là trình hoạt động (engine) và trình biên tập để viết lệnh, cho phép nhà phát triển có thể làm việc với các thành phần và các công cụ trợ giúp trực quan của Unity để từ đó có thể xây dựng trò chơi theo ý của nhà sản xuất. Engine và trình soạn thảo cung cấp:

* Một thành phần định danh NetworkIdentity cho các đối tượng của trò chơi trong mạng.
* Một trạng thái mạng có tên là NetworkBehaviour cho các kịch bản (script) viết để xây dựng hệ thống mạng.
* Đồng bộ hóa tự động các đối tượng chuyển đổi trong trò chơi.
* Tự động đồng bộ các biến script.
* Hỗ trợ đặt các đối tượng kết nối mạng trong cảnh Unity.
* Các thành phần mạng

(Các thành phần mà engine và trình soạn thảo cung cấp sẽ được nói rõ hơn ở các phần sau của tài liệu).

3.1.3 Các dịch vụ Internet của Unity

Unity cung cấp các dịch vụ Internet để hỗ trợ trò chơi của nhà phát triển trong suốt quá trình sản xuất và phát hành, bao gồm các dịch vụ:

* Dịch vụ kết nối
* Tạo các trận đấu (match) và quảng bá (advertise) các trận đấu.
* Liệt kê các trận đấu có sẵn và tham gia các trận đấu.
* Máy chủ chuyển tiếp
* Game chơi trên internet mà không có máy chủ chuyên dụng.
* Định tuyến các tin nhắn cho những người tham gia trận đấu.

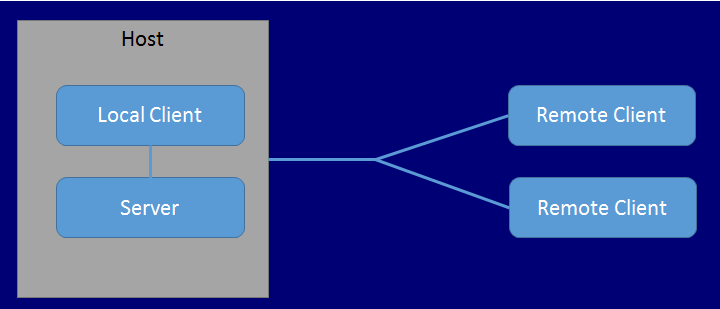
3.1.4 Network Transport và Network Layer

Lớp Transport Thời gian thực của Unity cung cấp:

* Tối ưu hóa UDP dựa trên các giao thức.
* Thiết kế đa kênh để tránh các vấn đề cổ chai (tắc nghẽn)
* Hỗ trợ nhiều mức chất lượng dịch vụ (QoS) trên mỗi kênh.
* Cấu trúc mạng linh hoạt hỗ trợ kiến trúc máy chủ ngang hàng hoặc mô hình kiến trúc máy khách-máy chủ.

3.2 Nguyên tắc hoạt động cơ bản của mạng đa người chơi nội bộ của Unity

Quá trình hoạt động của mạng lưới đa người chơi trên nền mạng LAN (mạng nội bộ) của Unity có thể được minh họa như sau:



Hình 3.2 Quy trình hoạt động của multiplayer networking[4]

Trong hệ thống kết nối mạng, trò chơi sẽ có một máy chủ và nhiều máy khách. Khi không có máy chủ chuyên dụng, một trong những máy khách đóng vai trò của máy chủ - Unity gọi máy khách này là "host".

Máy chủ của trò chơi vừa hoạt các quá trình, nhiệm vụ của một máy chủ mà nó lại vừa thực hiện các hoạt động của người chơi như một máy khách trong suốt cùng một quá trình là 1 trận đấu của trò chơi. Người chơi được chọn làm host sẽ là một loại máy khách đặc biệt, Unity gọi máy khách này là LocalClient, trong khi các máy khách còn lại sẽ được gọi là RemoteClients. LocalClient sẽ giao tiếp với máy chủ (địa phương) thông qua các cuộc gọi hàm trực tiếp và hàng đợi thông điệp, vì máy chủ của LocalClient nằm chung với nó trên cùng 1 máy tính, ngoài ra tuy rằng trên 1 máy thì cả 2 quá trình khách – chủ sẽ hoạt động song song trong cùng một quá trình là 1 trận đấu của trò chơi. Vì chung trên 1 máy nên LocalClient sẽ dễ dàng chia sẻ cảnh (trận đấu) với máy chủ. Còn các RemoteClients còn lại (người chơi khác) sẽ giao tiếp với máy chủ thông qua kết nối mạng thông thường.

Một trong những mục đích của hệ thống mạng là cho các mã (lệnh) của LocalClients và RemoteClients đều có thể hoạt động như nhau, để các nhà phát triển trò chơi chỉ phải tập trung suy nghĩ về một loại tập lệnh duy nhất trong suốt quá trình tạo ra trò chơi của mình.

3.2.1 Quá trình khởi tạo

Trong Unity, lệnh GameObject.Instantiate được dùng để tạo ra các đối tượng mới cho trò chơi. Nhưng với hệ thống mạng, các đối tượng cũng phải được "sinh ra" (spawn) để có thể hoạt động trên mạng. Quá trình spawn này chỉ có thể được thực hiện trên máy chủ, đồng thời máy chủ còn có nhiệm vụ đó là làm sao cho các đối tượng mới được tạo ra có thể tương tác được với các máy khách kết nối tới. Một khi các đối tượng được sinh ra, Spawning System sử dụng các nguyên tắc để quản lý vòng đời của các đối tượng phân tán và để quản lý các nguyên tắc đồng bộ hóa trạng thái của trò chơi.

Ngoài thực tế, 2 khái niệm Create (tạo ra) và Spawn (sinh ra) có vẻ có nghĩa gần giống nhau. Và trong Unity cũng vậy, lệnh gọi hàm Instantiate() dùng để Create (tạo ra) các đối tượng ta cũng thể tạm gọi nó là Spawn (sinh ra) các đối tượng. Tuy nhiên đối với mạng HLAPI từ "spawn" được sử dụng với ý nghĩa cụ thể hơn, chi tiết hơn. Trong mô hình có thẩm quyền của máy chủ HLAPI, để có thể "spawn" (sinh ra) một đối tượng trên máy chủ có thể được hiểu đó là quá trình chỉ có một mình máy chủ sẽ tạo ra các đối tượng của trò chơi cho các máy khách, và các đối tượng này sẽ được quản lý bởi hệ thống tái tạo (Spawning System). Còn Create là quá trình cũng tạo ra đối tượng của trò chơi nhưng điểm khác biệt đó là đối tượng đó chỉ do máy tạo ra quản lý và đối tượng đó không có tầm ảnh hưởng tới các máy khác trong hệ thống. Còn một điểm khác biệt giữa Create với Spawn đó là trong Spawning System, các cập nhật trạng thái sẽ luôn được gửi tới (cập nhật liên tục tại) các máy khách khi có một đối tượng nào đó thay đổi trên máy chủ. Ngoài ra, trong Spawning System, nếu có một đối tượng nào bị hủy trên máy chủ thì nó cũng sẽ bị hủy trên các máy khách. Các đối tượng được sinh ra bởi hệ thống sẽ được thêm vào danh sách các đối tượng trực tuyến mà máy chủ sẽ quản lý, do đó nếu có một khách khác tham gia trò chơi sau đó, các đối tượng sẽ dễ dàng được sinh ra trên máy khách đó từ danh sách đã được quản lý. Mỗi đối tượng này sẽ được quản lý theo định danh (Id) và mỗi định danh là duy nhất được gọi là "netId" sẽ được lưu trữ giống nhau trên máy chủ và máy khách. Việc này được sử dụng để có thể dễ dàng định tuyến các thông điệp tới các đối tượng và để xác định các đối tượng.

Khi các đối tượng được định danh được sinh ra trên máy khách, chúng sẽ được tạo ra với trạng thái hiện tại ánh xạ từ các đối tượng đó trên máy chủ. Tất cả những thông số, tham số, trạng thái hay di chuyển của các đối tượng được định danh sẽ được đồng bộ từ máy chủ cho tới các máy con (máy khách). Vì vậy các đối tượng nằm trên các máy khách sẽ luôn được cập nhật khi chúng được tạo ra. Điều này giúp hệ thống có thể tránh được các vấn đề mâu thuẫn. Ví dụ như, nếu các đối tượng không được quản lý chặt chẽ, sẽ có khả năng các đối tượng này sẽ được sinh ra ở một vị trí khởi tạo sai lệch so với vị trí được quy định, mà sau đó lại xuất hiện vị trí được chỉ định khi một gói cập nhật trạng thái đến, điều này khiến trò chơi trở nên khập khiễn, tạo độ trễ gây khó chịu cho người chơi.

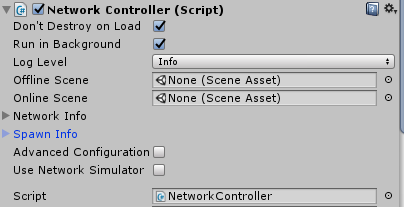
Để sinh ra các đối tượng trên client ta cần phải khởi tạo các đối tượng từ các prefab (model (mô hình (hình vẽ của đối tượng, các tham số, các phụ kiện đi kèm,…))) của đối tượng bằng cách truyền các mô hình này vào hàm NetworkServer.Spawn trên máy chủ. Để hỗ trợ kĩ thuật, Unity đã tạo ra các bảng panel xem trước của trình kiểm tra định danh mạng. Các bảng xem trước này cung cấp cho nhà phát triển các tham số về ID, nội dung của đối tượng được định danh, chính các giá trị này sẽ giúp client xác định prefab (tham số về vị trị, kích thước, các kịch bản (đoạn code),…) của đối tượng để từ đó có thể tạo các đối tượng. Để chương trình có thể hoạt động hiệu quả và ăn khớp, có một bước đăng ký mà tất cả các máy khách nào cũng phải thực hiện; đó là các máy khách sẽ phải gọi ClientScene.RegisterPrefab để nói (trao đổi) với hệ thống về nội dung của đối tượng mà máy khách muốn được sinh ra.

Quy trình khởi tạo diễn ra từng bước như sau:

* Định danh cho các đối tượng cần được khởi tạo và đăng kí các đối tượng này lên hệ thống
* Các đối tượng đã được đăng kí sẽ được khởi tạo từ prefab trên máy chủ
* Các mã lệnh của trò chơi dùng để khởi tạo các tham số, giá trị cho từng trường hợp (lưu ý rằng với các thành phần vật lý 3D của đối tượng sẽ không có hiệu lực ngay lập tức (đỗ trễ này rất ngắn))
* NetworkServer.Spawn() được gọi để tạo đối tượng
* Trạng thái của các SyncVars (các tham số chung của đối tượng dùng trên tất cả các máy) sẽ được khởi tạo trên máy chủ bằng cách gọi hàm OnSerialize() thuộc lớp NetworkBehaviour của Unity
* Một thông điệp mạng kiểu MsgType.ObjectSpawn được gửi đến các máy khách kết nối bao gồm các dữ liệu về các SyncVar
* Hàm OnStartServer() sẽ được gọi trên máy chủ, và biến isServer sẽ được gán thành true
* Máy khách nhận được thông điệp ObjectSpawn (lệnh (command) tạo đối tượng) và tạo một cá thể mới từ prefab đã đăng ký
* Các dữ liệu về SyncVar sẽ được áp dụng cho các cá thể (đối tượng) mới trên máy khách bằng cách gọi hàm OnDeserialize() thuộc lớp NetworkBehaviour của Unity
* Trên mỗi máy khách các đối tượng được tạo sẽ gọi hàm OnStartClient(), và isClient sẽ được đặt thành true
* Khi quá trình chơi diễn ra, các thay đổi đối với các giá trị SyncVar sẽ luôn được tự động đồng bộ hóa tới các máy khách.
* Nếu muốn hủy đối tượng, đầu tiên máy chủ sẽ gọi hàm NetworkServer.Destroy() để hủy đối tượng
* Một thông điệp mạng (command) có dạng MsgType ObjectDestroy được gửi tới các máy khách
* Khi nhận được lệnh, máy khách sẽ gọi hàm OnNetworkDestroy() để hủy đối tượng trên máy của mình.

Phần trên là quy trình khởi tạo các đối tượng ngoài phạm vi người chơi (các đối tượng không thuộc phạm vi người chơi điều khiển). Các đối tượng do người chơi điều khiển sẽ có một điểm cụ thể hơn một chút.

Ta sẽ tạo ra nhân vật của người chơi bằng công cụ NetworkManger (Network Controller – trình điều khiển mạng) của Unity.



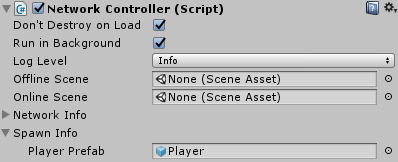
Hình 3.3 Thành phần công cụ NetworkManger của Unity

Sau đây là quy trình tạo nhân vật người chơi

* Tạo trình định danh cho Prefab với NetworkIdentity và đăng ký nhân vật vào ô Player Prefab thuộc mục Spawn Info của Network Controller



Hình 3.4 Trình định danh Network Identity



Hình 3.5 Đăng kí nhân vật của người chơi lên Network Controller

* Một khi máy khách kết nối tới máy chủ
* Máy khách sẽ gọi hàm AddPlayer(), sau đó máy khách sẽ gửi một thông điệp mạng có kiểu MsgType.AddPlayer lên máy chủ
* Máy chủ nhận được thông điệp và thực hiện các cuộc gọi tới hàm NetworkManager.OnServerAddPlayer()
* Nhân vật sẽ được khởi tạo từ Player Prefab trên máy chủ
* NetworkManager.AddPlayerForConnection() để đưa nhân vật vừa được tạo trên máy chủ vào đường truyền kết nối
* Vì ta đã đưa nhân vật vào đường truyền kết nối nên ta không cần phải gọi hàm NetworkServer.Spawn()
* Một thông điệp mạng kiểu MsgType.Owner sẽ được gửi đến máy khách để thông báo rằng nhân vật đã được truyền đi (máy nào cần tạo nhân vật thì chỉ có máy đó mới nhận được thông điệp)
* Máy khách đã yêu cầu ở ban đầu sẽ nhận được thông điệp mạng
* Máy khách sẽ gọi hàm OnStartLocalPlayer(), tạo nhân vật, điều khiển nhân vật và biến isLocalPlayer được đặt thành true. Kết thúc quá trình người chơi đã có riêng cho mình một nhân vật để điều khiển

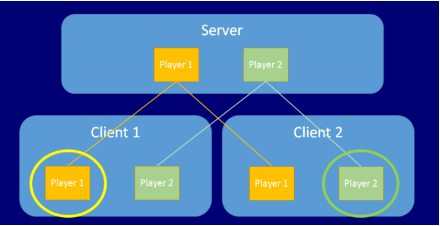
Nếu có xảy ra tranh chấp trong việc tạo dựng đối tượng thì ta cần lưu ý rằng hàm OnStartLocalPlayer() chỉ được gọi sau khi đã được gọi và thực thi hàm OnStartClient(), cũng giống như quá trình máy chủ (server) chỉ gửi các thông báo điệp Owner sau khi đối tượng người chơi đã được sinh ra. Mục đích của việc làm này đó là để tránh biến isLocalPlayer không bị ảnh hưởng khi gọi hàm OnStartClient().

isLocalPlayer là một biến để phân biệt nhân vật giữa máy tính này (người chơi này) với nhân vật của các máy khác (các người chơi khác) trong hệ thống.

Ta có thể sử dụng hàm OnStartLocalPlayer() để thực hiện khởi tạo các giá trị liên quan đến máy khách. Điều này có thể bao gồm việc cho phép xử lý các dữ liệu đầu vào và cho phép theo dõi khung nhìn (camera) đối tượng của người chơi. Thông thường chỉ có máy khách của người chơi mới có thể kích hoạt cho camera hoạt động.

3.2.2 Người chơi, người chơi địa phương và sự ủy quyền

Trong hệ thống mạng, các đối tượng người chơi là các đối tượng đặc biệt. Mỗi người chơi sẽ điều khiển cho riêng mình một đối tượng (nhân vật), và hệ thống mạng sẽ tương tác với các đối tượng đó thông qua các lệnh và thông điệp. Mỗi người chơi chỉ có thể điều khiển nhân vật riêng của mình và không thể điều khiển các nhân vật khác của những người chơi khác. Vì vậy, ta có thể bảo các đối tượng (nhân vật) người chơi có tính "độc nhất ". Khi một người chơi mới tham gia vào trò chơi và hệ thống thực hiện tạo kết nối cho người chơi tới hệ thống hoàn tất, đối tượng người chơi đó sẽ trở thành đối tượng "local player" (người chơi địa phương) trên máy khách của người chơi đó (nghĩa là chỉ có máy khách đó quản lý người chơi đó). Lúc đó, thuộc tính isLocalPlayer sẽ được đặt thành true, và hàm OnStartLocalPlayer() sẽ được gọi trên đối tượng trên máy khách. Hình sau sẽ minh họa mối quan hệ giữa người chơi, máy chủ và máy khách.

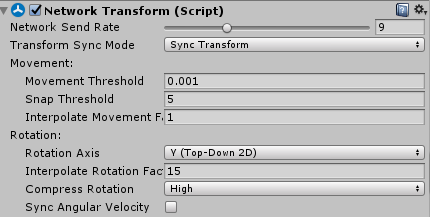


Hình 3.6 Mối quan hệ giữa player, client và server[4]

Ta thấy trận đấu (trò chơi) của ta xảy ra ở đây có 2 người chơi tham gia (là Player 1 và Player 2) tương ứng với 2 máy khách là Client 1 và Client 2. Trong đó Client 1 (người chơi thứ nhất) sẽ điều khiển nhân vật có tên là Player 1. Tương tự người chơi thứ 2 (Client 2) sẽ điều khiển nhân vật có tên là Player 2. Ở mỗi Player khi khởi tạo sẽ được gán thêm một cờ (biến) có tên là isLocalPlayer dùng để chương trình xác định xem người chơi đang điều khiển Player nào. Biến isLocalPlayer thuộc kiểu boolean có 2 giá trị duy nhất là true hoặc false dùng để đánh dấu. Player do người chơi điểu khiển thì isLocalPlayer sẽ có giá trị là true. Và Player không do người chơi điều khiển sẽ được gán giá trị là false. Như vậy, ở Client 1 thì isLocalPlayer ở Player 1 sẽ có giá trị là true, còn isLocalPlayer ở Player 2 sẽ có giá trị là false. Tương tự với Client 2.

Mục đích của cờ (biến) isLocalPlayer này đó là giúp chương trình có thể phân biệt người chơi điều khiển nhân vật (Player) nào, để từ đó ta có lọc riêng khung nhìn, bảng điều khiển đi lại, phím bắn,… chỉ riêng cho Player đó thôi mà không làm ảnh hưởng tới các Player khác trong hệ thống.

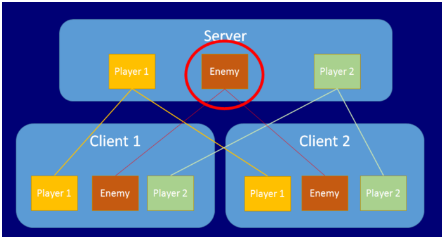
Mục đích của hệ thống đa người chơi đó là giúp những người chơi có tương tác được với nhau. Vì vậy việc đồng bộ hóa các trạng thái của trò chơi cho tất cả người chơi là một điều quan trọng. Vì nếu không đồng bộ hóa, Player 1 sẽ có thể không “thấy” hình ảnh của Player 2 để từ đó có thể tương tác. Để chương trình có thể đồng bộ hóa cho tất cả người chơi, việc đơn giản ta sẽ làm đó là thêm vào đối tượng Player công cụ đồng bộ mạng NetworkTransform của Unity.



Hình 3.7 Công cụ Network Transform của Unity

Đối với các đối tượng không thuộc phạm vi của người chơi ví dụ như kẻ thù, đồng minh trí tuệ nhân tạo,… hay nói cách khác các đối tượng này thuộc sự quản lý của trò chơi, do đó quyền lực điều khiển các đối tượng này sẽ nằm trên máy chủ.

Hình sau mô tả mối liên hệ giữa Client, Player, Enemy và Server.



Hình 3.8 Mối quan hệ giữa Client, Player, Server và đối tượng không thuộc người chơi[4]

Trong lớp NetworkBehaviour có một thuộc tính có tên là "hasAuthority" có thể được sử dụng để ủy quyền cho một đối tượng. Khác với nhân vật của người chơi, các đối tượng không thuộc người chơi hay nói cách khác không phải do người chơi điều khiển nên ta cần phải ủy quyền các đối tượng này cho server điều khiển.

3.2.3 Sự ủy quyền cho các đối tượng không phải là người chơi

Khi ủy quyền cho một máy khách hay nói cách khác đó là ta gọi hàm OnStartAuthority() được gọi từ NetworkBehaviours trên đối tượng, và thuộc tính hasAuthority sẽ được gán là true. Trên các máy khách khác do không phải các máy này điều khiển thì thuộc tính hasAuthority sẽ là false. Các đối tượng không phải là người chơi đã được ủy quyền cho máy khách cũng có thể gửi lệnh (command), giống như các đối tượng người chơi. Các lệnh mà các đối tượng không thuộc người chơi gửi sẽ được chạy trên máy chủ của đối tượng, điều này giúp đảm bảo không có bất kì một người chơi nào có thể liên quan đến kết nối.

Các đối tượng không phải là người chơi được sự ủy quyền của máy khách phải được LocalPlayerAuthority kiểm tra trong công cụ NetworkIdentity của các đối tượng này.

3.2.4 Các tính chất phạm vi của hệ thống mạng

Trong lớp NetworkBehaviour có các thuộc tính cho phép kịch bản (script) biết ngữ cảnh mạng nào thuộc về đối tượng. Các ngữ cảnh này bào gồm:

* isServer - true nếu đối tượng thuộc phạm vi điều khiển bởi máy chủ (hoặc host) và đã được sinh ra.
* isClient - true nếu đối tượng thuộc phạm vi điều khiển bởi máy khách, và đã được tạo ra bởi máy chủ.
* isLocalPlayer - true nếu đối tượng là một đối tượng thuộc phạm vi điều khiển bởi người chơi do máy khách của người chơi quản lý.
* hasAuthority - true nếu đối tượng thuộc phạm vi điều khiển bởi máy khách của người chơi

Các thuộc tính này có thể được gán tùy chỉnh giá trị true hoặc false tùy vào nhà phát triển trong các đoạn script bằng các câu lệnh.

3.3 Network Manger

Để đưa nền tảng mạng (network) vào trò chơi, trước hết ta cần thiết lập một trình điều khiển mạng cho nó. Để thiết lập ta làm theo những bước sau:

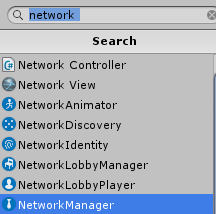
Bước 1, tạo một game object (đối tượng trò chơi) rỗng.



Hình 3.9 Tạo một game object rỗng

Bước 2, sửa tên game object rỗng thành Network Manger

Bước 3, thêm công cụ Network Manager vào đối tượng



Hình 3.10 Thêm công cụ NetworkManager vào đối tượng

Vậy là ta đưa nền tảng mạng thành công vào trò chơi.

3.4 Thiết lập prefab nhân vật của người chơi

Như đã nói ở các phần trước, nhân vật của người chơi ta không thể tạo trực tiếp trên bản đồ vì có 2 lý do:

* Một là, ta không thể biết được chính xác có bao nhiêu đối tượng người chơi tham gia vào trò chơi ở mọi thời điểm mà trò chơi đang hoạt động.
* Hai là, nếu tạo trực tiếp như vậy, trình điều khiển mạng sẽ khó quản lý được đối tượng.

Vì vậy ta sẽ phải thiết lập một prefab (mô hình) đối tượng nhân vật của người chơi, từ mô hình này trình điều khiển sẽ tạo ra (lưu ý là tạo ra ở đây có nghĩa là “Spawn”) nhân vật cho người chơi điều khiển.

Prefab của nhân vật ở đây sẽ gồm các thành phần:

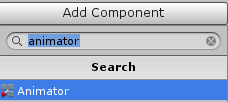
* Mô hình của nhân vật (hình ảnh của nhân vật) trong trò chơi
* Bộ quản lý (trình quản lý) chuyển động của nhân vật (Animator)
* Trình quản lý va chạm theo mẫu khối (Box Collider)
* Các đoạn scipt quản lý hoạt động, camera, VR, âm thanh, vũ khí,… của đối tượng
* Âm thanh cho đối tượng (Audio Source)
* Trình quản lý các hiệu ứng vật lý (trọng lực (gravity),…) (RigidBody)
* Định danh mạng cho đối tượng (Network Identity)
* Trình đồng bộ mạng (Network Transform)

**Bước 1**, ta kéo mô hình nhân vật ta đã vẽ và lưu từ assets lên bản đồ (scence)



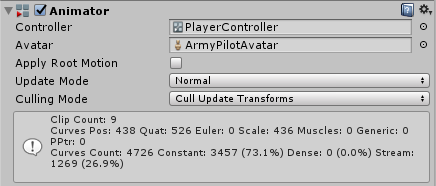
Hình 3.11 Tạo mô hình thô cho nhân vật

**Bước 2**, Bấm Add Component và thêm trình điều khiển hoạt động cho nhân vật (Animator)



Hình 3.12 Thêm trình điều khiển hoạt động cho nhân vật

Gán vào Controller của Animator bộ quản lý hoạt động mà ta đã tạo



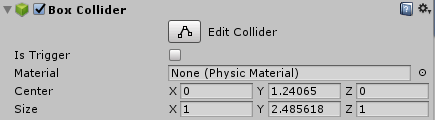
Hình 3.13 Animator sau khi thiết lập

**Bước 4**, kéo các đoạn script ta đã viết vào đối tượng



Hình 3.14 Các đoạn script đã được kéo vào

**Bước 5**, thêm Box Collider



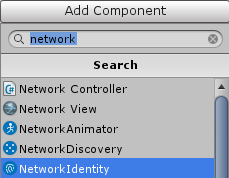
Hình 3.15 Box Collider sau khi được thêm vào nhân vật

**Bước 6**, thêm Audio Source (âm thanh) cho nhân vật



Hình 3.16 Sau khi thêm Audio Source vào nhân vật

**Bước 7**, thêm trình định danh cho nhân vật (Network Identity)



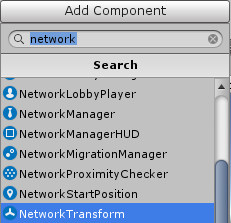
Hình 3.17 Thêm NetworkIdentity vào nhân vật

Vì đây là đối tượng do người chơi hay máy khách quản lý nên ta chọn mục Local Player Authority.



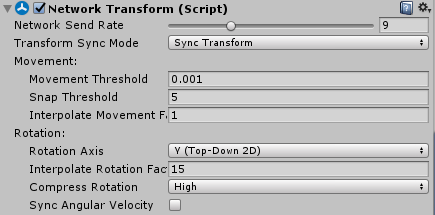
Hình 3.18 Ủy quyền cho máy người chơi quản lý nhân vật

**Bước 8**, thêm trình đồng bộ hóa vào nhân vật



Hình 3.19 Thêm trình đồng bộ hóa vào nhân vật

Chỉnh các tham số của trình đồng bộ theo hình sau:



Hình 3.20 Trình đồng bộ hóa của nhân vật

**Bước 9**, thêm RegidBody vào nhân vật

**Bước 10**, thêm camera và công cụ VR vào nhân vật (sẽ nói rõ hơn ở chương 4).

3.5 Viết Script mạng cho nhân vật

3.5.1 Chuyển từ script thông thường sang script mạng

Để chuyển từ một script không mạng thành script có thể chạy trên nền tảng mạng, đầu tiên, ta sử dụng thư viện UnityEngine.Networking.

Sau đó ta cho lớp của Script kế thừa sang lớp NetworkBehaviour.

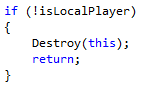
Ta có minh họa như sau:



Hình 3.21 Cách chuyển từ 1 script thường sang script mạng

3.5.2 Lọc bỏ các thành phần không thuộc người chơi

Để máy khách (client) có thể biết được nên lọc bỏ các đối tượng không thuộc phạm vi của người chơi mình đang tham gia dùng hàm if(!isLocalPlayer) như sau:

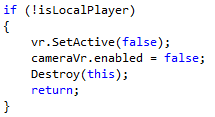


Hình 3.22 Đoạn code lọc bỏ các đối tượng không thuộc người chơi

Nếu các đối tượng nào không thuộc phạm vi trực tiếp của người chơi ta sẽ dùng hàm Destroy(this) để xóa script này nếu script này không phải là script của nhân vật của người chơi đang tham gia.

Với đoạn script ta sẽ lọc bỏ được các đoạn script không trực thuộc nhân vật của người chơi đang tham gia giúp máy của người chơi có thể tập trung điều khiển vào nhân vật của người chơi giúp tránh được tình trạng người chơi này điều khiển một thành phần nào đó của người chơi khác. Nói chung là giúp nhân vật và người chơi thống nhất với nhau trong thể.

Tận dụng điều này ta có thể tắt các camera của các người chơi để trò chơi tập trung vào camera vào nhân vật mà người chơi của client đang sở hữu. Ta thực hiện bằng cách gọi lệnh Camera.enable = false (Camera trong minh họa sau sẽ có tên biến là cameraVr):



Hình 3.23 Tắt các camera của các người chơi khác

Như vậy sau khi thực hiện các đoạn script trên, mỗi người chơi chỉ có điều khiển nhân vật cũng như chỉ thấy được những gì mà nhân vật mình đang thấy. Điều này khiến trò chơi trở nên logic và mạch lạc hơn.

3.6 Quản lý vật thể ngoài phạm vi của người chơi trong hệ thống mạng

3.6.1 Các hàm quản lý trạng thái trực tuyến

Có bốn hàm giúp ta có thể quản lý trực tuyến đó là:

* OnStartClient
* OnStartHost
* OnStopClient
* OnStopHost

Để sử dụng 4 hàm này ta vẫn sẽ phải khai báo thư viện UnityEngine.Networking và kế thừa lớp NetworkBehavious của Unity.

Ta có, OnStartHost sẽ chạy những đoạn code trong nó khi có người chơi host (máy chủ) xuất hiện.

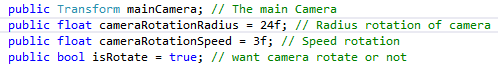
Tương tự, OnStartClient sẽ chạy những đoạn code trong nó trên máy của người chơi chỉ khi người chơi tham gia vào trận đấu trực tuyến.

Ngược lại với OnStartHost thì OnStopHost sẽ chạy những đoạn code trong nó khi người chơi host chấm dứt và thoát ra ngoài hệ thống.

Tương tự OnStopClient sẽ chạy những đoạn code trong nó khi người chơi thoát ra khỏi trận đấu.

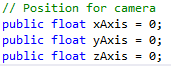
Để minh họa, ta sẽ làm một ứng dụng đơn giản đó là ta sẽ tạo một khung nhìn gợi ý về bản đồ mà người chơi sẽ tham gia. Khung nhìn này sẽ quay toàn cảnh 360 độ một phần nào đó của bản đồ.

Đầu tiên, ta sẽ khai báo đối tượng Camera, bán kính mà Camera sẽ quay, và tốc quay của nó. Đồng thời ta sẽ khai báo một cờ có biến logic (bool) để xác định lúc nào thì camera sẽ được bật.



Hình 3.24 Khai báo Camera và các thuộc tính liên quan

Tiếp tục ta sẽ khai báo vị trí bắt đầu quay của camera



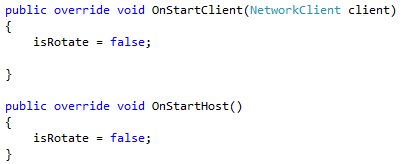
Hình 3.25 Khai báo vị trí bắt đầu quay của camera

Cuối cùng là khai báo góc quay của nó



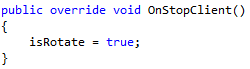
Hình 3.26 Khai báo góc quay của của camera

Ta chỉ cho camera quay quanh bản đồ khi người chơi ở ngoài trận đấu vì vậy trong trận đấu ta sẽ phải tắt camera này đi. Vì vậy ta sẽ chỉnh cờ isRotate = false để thông báo chương trình tắt camera này đi bằng cách đặt lệnh này trong 2 hàm là OnStartClient và OnStartHost.



Hình 3.27 Bật cờ thông báo hủy camera

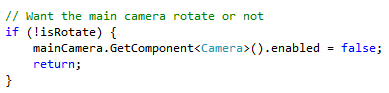
Còn nếu khi người chơi thoát ra khỏi trận đấu ta sẽ bật camera này quay bình thường bằng cách đặt cờ isRotate = true; trong hàm OnStopClient.



Hình 3.28 Thông báo bật camera khi người chơi thoát khỏi trận đấu

Để camera có thể quay liên tục hoặc bật tắt bất cứ lúc nào khi trò chơi diễn ra, ta sẽ tiếp tục viết code cho hàm Update().

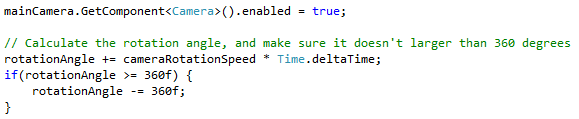
Trong hàm Update, đầu tiên, ta sẽ xét xem chương trình muốn camera bật hay tắt. Ta sẽ thực hiện bằng cách xét xem biến isRotate là true hay false. Nếu là false nghĩa là chương trình muốn tắt camera. Ta sẽ tắt camera bằng các lệnh sau:



Hình 3.29 Tắt camera nếu cờ isRotate là false

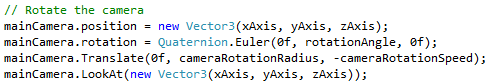
Nếu biến isRotate = true ta sẽ thực hiện các lệnh sau:

Bật camera và tạo góc xoay cho camera (nếu góc xoay camera lớn hơn 360 thì cho góc xoay quy về 0).



Hình 3.30 Khởi động camera và tạo góc xoay cho camera

Và công đoạn cuối cùng đó là xoay camera và dịch chuyển camera cùng một lúc.



Hình 3.31 Xoay và dịch chuyển camera

Đầu tiên ta sẽ lấy vị trí hiện tại của camera. Sau đó ta sẽ thực hiện phép xoay bằng cách gọi hàm Quaternion.Euler() là phép dịch quaternion trong toán học vì vậy ta cần ba tham số là x, y, và z ứng với mỗi trục của đồ thị. Do là phép xoay tròn nên x và z không đổi và xoay quanh tâm nên x và z ta truyền 0, còn y ta sẽ truyền vào là góc xoay mà ta đã tính được ở trên.

Tiếp tục ta sẽ thực hiện dịch camera bằng hàm Translate() với ba tham số tương ứng là x, y, z là 3 trục của tọa độ.

Còn LookAt là hàm ta muốn Camera nhìn về điểm nào trên bản đồ.

Như vậy là ta đã hoàn thành xong ứng dụng áp dụng 4 hàm quản lý trạng thái mạng của Unity.

3.6.2 Quản lý các đối tượng khác ngoài phạm vi người chơi

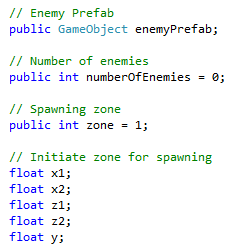
Để hệ thống mạng quản lý các đối tượng khác ngoài nhân vật của người chơi, ta cũng phải thêm cho đối tượng trình định danh. Tuy nhiên ta sẽ không chọn Local Player Authority vì các đối tượng này không phải do người chơi quản lý, do đó ta sẽ chọn là Server Only nghĩa là để server quản lý.



Hình 3.32 Định danh mạng cho đối tượng và ủy quyền cho server quản lý

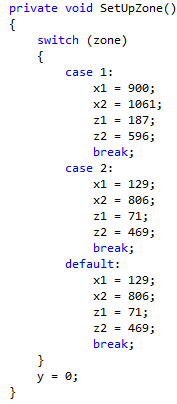
Để áp dụng, ta sẽ tạo ứng dụng tạo kẻ thù (zombie) một cách tự động khi trò chơi bắt đầu.

Đầu tiên ta sẽ khai báo các tham số cho script bao gồm: mô hình zombie, số lượng zombie muốn thả lên bản đồ, tên khu vực muốn thả và kích thước của khu vực.



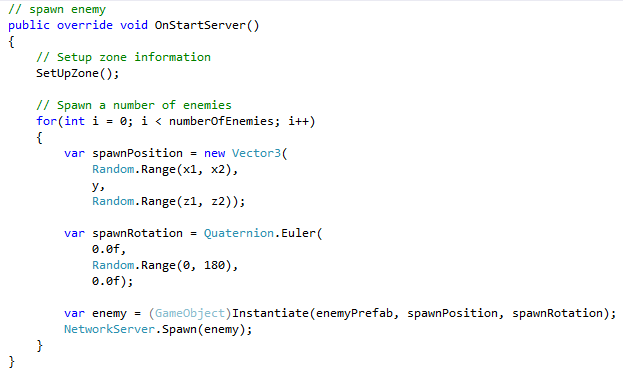
Hình 3.33 Khởi tạo các thuộc tính cho lớp tạo zombie

Hàm thiết lập khu vực:



Hình 3.34 Hàm thiết lập khu vực

Zombie ta đã ủy cho server quản lý nên khi server được khởi động ta có thể tạo tự động (spawn) các đối tượng này. Hàm OnStartServer() sẽ chạy những lệnh trong nó khi server được kích hoạt. Khi server được kích hoạt, ta sẽ cho chạy hàm SetUpZone() để thiết lập tham số cho khu vực zombie sẽ được tạo. Tiếp theo ta sẽ cho chạy vòng lặp để tạo ra đúng số zombie yêu cầu. Ở mỗi vòng lặp ta sẽ thực hiện tạo zombie bằng cách dùng hàm Instance() để tạo ra mô hình cho zombie và hàm Spawn() để làm zombie xuất hiện trên bản đồ theo đúng các tham số mà ta đã truyền vào khi tạo mô hình cho zombie.



Hình 3.35 Hàm tạo zombie tự động khi trận đấu mạng bắt đầu

3.7 Xây dựng hành động bắn trực tuyến

Ở chương 2, ta biết cách xây dựng để nhân vật có thể bắn ra đạn, nhưng tuy nhiên việc bắn đạn này chỉ xảy ra ở máy tính của người chơi mà thôi, các máy tính khác của những người chơi sẽ không thể thấy được điều này. Vì vậy ta sẽ chỉnh sửa để đưa hành động này có thể xảy ra trên mạng.

Đầu tiên sẽ thêm thư viện UnityEngine.Networking cho lớp PlayerFire\_Network (lớp chứa hành động bắn) và lớp BulletController (lớp quản lý đạn).



Hình 3.36 Thêm thư viện UnityEngine.Networking cho lớp

Tiếp đó, để đưa các đoạn script có thể chạy trên mạng, ta sẽ cho lớp PlayerFire\_Network và BulletController kế thừa lớp MonoBehaviour (chạy trên máy tính đơn) sang kế thừa lớp NetworkBehaviour (chạy trên nhiều)



Hình 3.37 Chuyển sang kế thừa lớp NetworkBehaviour

Quá trình bắn đạn này gồm 2 phần: phần 1, client sẽ gửi lệnh command yêu cầu server xuất đạn; và phần 2 là phần server sẽ phản hồi yêu cầu client destroy đạn sau một khoảng thời gian.

3.7.1 Thiết lập công cụ mạng cho mô hình đạn

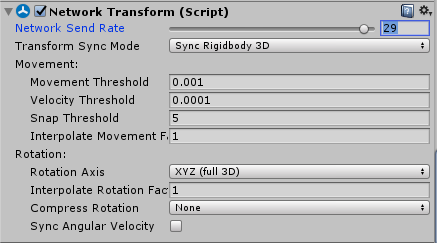
Để mô hình có thể hoạt động trên mạng ta cần phải đưa nó vào danh sách quản lý của trình điều khiển mạng. Trình điều khiển chỉ quản lý các đối tượng có trình định danh mạng (NetworkIdentiy), vì vậy ta sẽ thêm NetworkIdentity vào mô hình đạn như sau.



Hình 3.38 Thêm NetworkIdentity cho mô hình Bullet

Do mô hình đạn chịu sự thực thi của cả 2 phía là server và client nên ta sẽ không chọn cho đối tượng chịu sự quản lý duy nhất của ai cả. Ta không chọn tức là nó sẽ được chịu sự quản lý của cả 2 là server và client.

Để các máy tính khác có thể thấy đạn bắn ra, ta cần thêm trình đồng bộ mạng (NetworkTransform) cho đối tượng.



Hình 3.39 Thêm NetworkTransform

Network Send Rate là số frame sẽ xảy ra trong 1s của trò chơi mà hệ thống có thể gửi đi. Vì đạn bay nhanh nên ta sẽ chỉnh Network Send Rate lên cao.

Cuối cùng ta sẽ thêm đối tượng mô hình đạn vào danh sách các đối tượng sẽ được sinh ra bởi hệ thống.

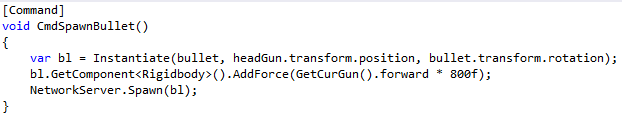


Hình 3.40 Thêm Bullet vào Registered Spawnable Prefabs

3.7.2 Viết lệnh Command cho client

Sau khi thiết lập các thành phần mạng xong cho mô hình đạn, ta sẽ viết lệnh Command để client có thể yêu cầu server sinh đạn bắn.

Ta có đoạn lệnh như sau:



Hình 3.41 Cấu trúc hàm sinh đạn trên hệ thống mạng

Trước hàm ta sẽ đề chữ [Command] cho chương trình biết đây là lệnh sẽ được gửi lên server.

Instantiate() là hàm khởi tạo đối tượng bullet bắt đầu từ vị trí headGun.transform.position và độ xoay xuất phát từ bullet.transform.rotation.

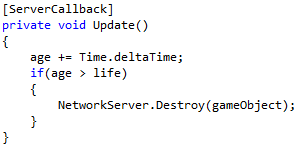
AddForce() là hàm tạo lực giúp tạo một lực đẩy cho đạn khi bắn ra khỏi nòng súng (headGun).

Và cuối cùng là NetworkServer.Spawn(bl); chính là lệnh mà client yêu cầu server xuất ra trò chơi mô hình đạn đã được ta khởi tạo.

3.7.3 Viết lệnh ServerCallback của server

Để tránh gây tắc mạng khi quản lý nhiều đối tượng và cũng như tránh mô hình đạn chịu sự chỉ quản lý từ 1 bên (do ta không setup ở 3.7.1), ta sẽ giải phóng mô hình đạn sau một gian khi nó đã được bay đi.

Ta có đoạn lệnh sau:



Hình 3.42 Cấu trúc hàm giải phóng đạn ra khỏi mạng và xóa mô hình đạn

Trước hàm ta sẽ đề chữ [ServerCallback] để thông báo cho chương trình biết đây là lệnh sẽ được server gửi xuống client.

Age là một biến đếm xem đạn đã tồn tại trên hệ thống đã được bao lâu.

Nếu thời gian tồn tại lớn hơn thời gian ta cho phép nó tồn tại (life = 2.0 s) thì ta sẽ giải phóng và tiêu hủy nó.

Cuối cùng ta có lệnh NetworkServer.Destroy(gameObject) là lệnh server sẽ gửi xuống cho tất cả client hãy xóa và giải phóng mô hình đạn này ra khỏi hệ thống.

Như vậy là quá trình bắn đạn trên hệ thống mạng đã hoàn tất.

3.8 Tổng kết chương

Sau chương này ta đã biết một hệ thống mạng do Unity xây dựng sẽ hoạt động như thế nào. Cách nó quản lý đối tượng cũng như là cách nó đồng bộ các đối tượng trên các máy tính trong hệ thống của mình.

Ta cũng biết được tổng quát các framework mạng của Unity.

Đồng thời ta cũng biết được cách xây dựng các chức năng để nó có thể xảy ra trên hệ thống mạng (sinh zombie, bắn đạn,…).

CHƯƠNG 4 – XÂY DỰNG VR CHO TRÒ CHƠI

Unity có các công cụ để hỗ trợ cho việc phát triển các trò chơi có liên quan đến việc sử dụng các thiết bị VR. Chương này sẽ tập trung vào phát triển trò chơi có thể hoạt động trên nền tảng các thiết bị VR của Oculus, đặc biệt là bộ VR Oculus Rift Development Kit 2 (DK2) và phiên bản VR dành cho người tiêu dùng của Gear VR (một loại thiết bị VR có thể chạy trên các loại điện thoại di động như Samsung Galaxy S6, S6 Edge, S6 Edge + hoặc Note 5,…).

4.1 Kiến nghị về phần cứng và phần mềm

Vì VR là một công nghệ mới nên có thể ta sẽ cần có một số kiến nghị về phần cứng và phần mềm để chương trình có thể hoạt động với một hiệu suất thật hiệu quả. Lưu ý đây chỉ là các kiến nghị chứ không phải là các yêu cầu.

4.1.1 Kiến nghị về phần cứng

Yêu cầu cao về frame rate của HMD (Head-Mounted Display – thiết bị đeo trên đầu (VR)) là điều cần thiết để người dùng có một trải nghiệm VR thoải mái, và frame rate (tần suất khung nhìn) của HMD phải làm sao cho thật phù hợp với refresh rate (tần suất làm mới) của bảng điều khiển được sử dụng trong HMD. Trong DK2, tốc độ frame rate này phải là 75 khung hình / giây và đối với Gear VR thì tốc độ này phải là 60 khung hình / giây. Nếu tỷ lệ khung hình giảm xuống dưới mức này, các hiệu ứng hình ảnh có thể sẽ khiến thị lực cũng như não bộ trở nên ảnh hưởng tới sức khỏe (đau đầu, buồn nôn,…).

Trong khi VR cố gắng đạt được các tỷ lệ khung hình yêu cầu, thì các GPU trong các máy tính cá nhân mà VR đang chạy trên nó cũng phải có khả năng xuất ra độ phân giải yêu cầu tương xứng với refresh rate của HMD. Trong trường hợp của DK2, thì refresh rate yêu cầu cho máy tính là 75 Hz với kích thước khung nhìn là 1920 x 1080. Lưu ý rằng nếu ta định phát triển trên các thiết bị Oculus Rift thương mại (CV1), độ phân giải này phải có độ phân giải là 2160 x 1200 ở 90hz, điều này đòi hỏi nhiều hơn so với DK2.

Nếu ta đang sử dụng DK2, hãy đảm bảo rằng phần cứng của ta có khả năng xuất ra độ phân giải yêu cầu theo refresh rate cần thiết. Nhiều máy tính xách tay sử dụng chipset để chuyển đổi giữa các chipset rời và tích hợp, và các chipset này thường không thể xuất ra 1920 x 1080 ở 75hz, dẫn đến một trải nghiệm VR tồi tệ cho người chơi.

4.1.2 Kiến nghị về phần mềm

Yêu cầu về OS X: Vào thời điểm này, có thể phát triển trên OSX 10.9+ với bộ chạy là Oculus 0.0.5 nhưng khi Oculus tạm dừng phát triển cho OS X, nhà phát triển trò chơi nên sử dụng Windows cho các chức năng VR gốc trong Unity.

Yêu cầu về Windows: Windows 7, 8, 8.1 và Windows 10 đều tương thích.

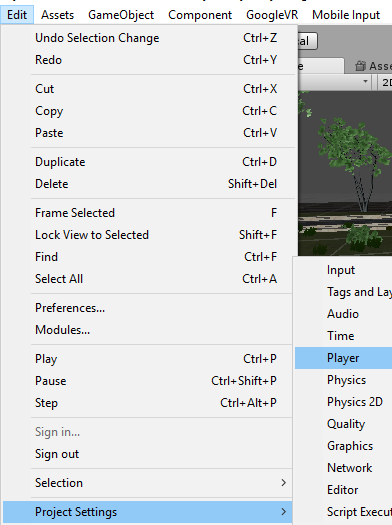
Yêu cầu về Android: Nhà phát triển nên sản xuất các trò chơi chạy trên nền tảng từ Android OS Lollipop 5.1 trở lên.

Trình điều khiển thẻ đồ họa (Graphic Card Driver): Các trình điều khiển này ta nên cần phải thường xuyên cập nhật. Các trình điều khiển cũ có thể không được hỗ trợ.

Oculus Runtime: Để sử dụng bộ hỗ trợ VR trong Unity, ta nên sử dụng Oculus Runtime 0.8 hoặc cao hơn.

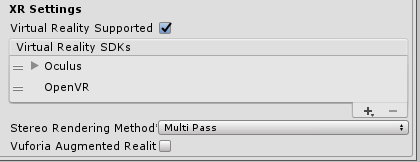
4.2 Thiết lập VR cho trò chơi

Để cho trò chơi ta sẽ thiết lập cho nhân vật của trò chơi được phép có sự hỗ trợ từ VR bằng cách, ta vào Edit chọn Project Settings chọn Player.



Hình 4.1 Thiết lập cho nhân vật hỗ trợ VR

Ở mục XR Settings, ta bấm chọn Virtual Reality Supported



Hình 4.2 Bấm chọn chế độ hỗ trợ VR cho nhân vật

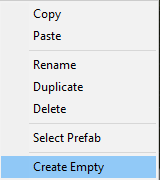
Như vậy là ta đã thiết lập xong cho trò chơi.

4.3 Thiết lập VR cho nhân vật

Để sử dụng VR cho nhân vật ta sẽ phải tải bản SDK của hãng hỗ trợ driver cho các nhà sản xuất thiết bị VR. Ta sẽ tải bản SDK của GoogleVR.

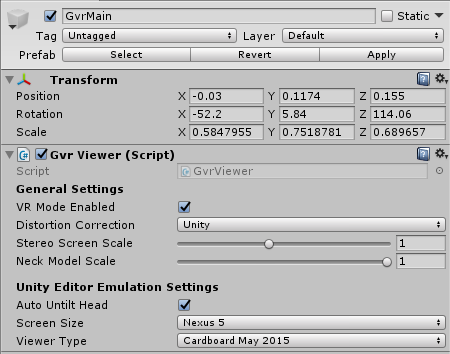
4.3.1 Thêm các đoạn script VR cơ bản

Sau khi tải bản SDK VR về xong ta sẽ tạo một game object rỗng có tên là GvrMain.



Hình 4.3 Tạo một game object rỗng

Sau đó trên game object này ta sẽ thêm SDK vừa mới tải về vô.

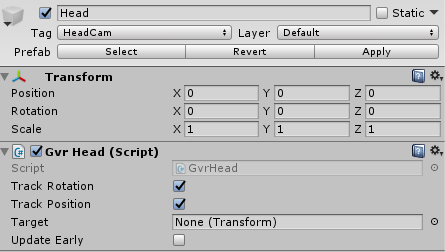


Hình 4.4 Thêm VR vừa tải vô game object của player

Đoạn script ta vừa mới thêm ở trên có chức năng:

* Truy vấn thiết bị để xem thông số
* Truy xuất dữ liệu lưu lại được ở thời điểm gần nhất của người chơi
* Cung cấp cảnh hiển thị cho thiết bị để chỉnh sửa nếu hình ảnh bị méo (tùy chọn - optional)

Tiếp tục ta sẽ tạo game object rỗng có tên là Head. Và thêm vào đoạn script sau:



Hình 4.5 Thêm script GvrHead vào Head

GvrHead có chức năng: Tập lệnh này cung cấp hỗ trợ theo dõi đầu cho máy ảnh. Đính kèm tập lệnh này với bất kỳ đối tượng trò chơi nào phù hợp với chuyển động đầu của nhân vật người chơi. Theo mặc định, nó liên tục cập nhật biến đổi cục bộ sang GvrViewer.HeadView. Một đối tượng đích có thể được chỉ định để cung cấp một khung tham chiếu thay thế cho chuyển động.

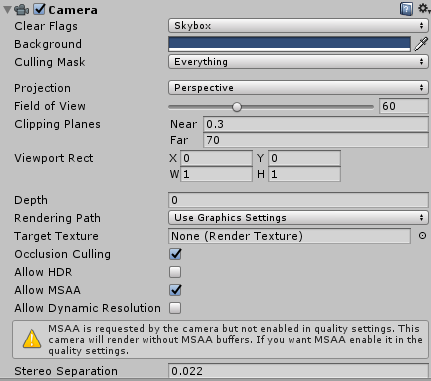
Trong một số trường hợp, ta cần VR cho hai trường hợp của GvrHead, đề cập đến hai mục tiêu khác nhau (một trong số đó có thể là camera. Sử dụng thuộc tính #trackRotation và #trackPosition trong trường hợp này.

4.3.2 Tạo camera VR cho người chơi

Trên đầu của người chơi ta sẽ tạo và gắn 1 game object rỗng. Đổi tên game object thành Main Camera.

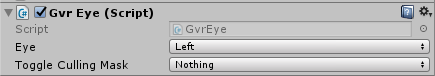
Trong Main Camera ta sẽ tạo thêm 4 game object có tên lần lượt là: Main Camera Left (mắt trái của VR), Main Camera Right (mắt phải của VR), GvrReticalPointer (tâm nhìn của VR) và pointerswalid (quản lý con trỏ trên màn VR của người chơi).

Trong Main Camera ta sẽ thêm thành phần Camera

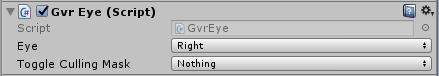


Hình 4.6 Thêm thành phần Camera cho Main Camera

Đối với Main Camera Left và Main Camera Right ta lần lượt thêm vào 2 object 2 đoạn script tương ứng sau:

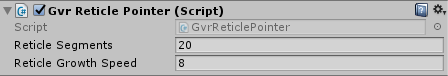


Hình 4.7 Thêm Gvr Eye cho Main Camera Left



Hình 4.8 Thêm Gvr Eye cho Main Camera Right

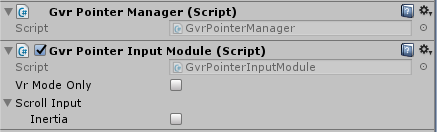
Cuối cùng là ta thêm vào Script tâm nhìn cho VR:



Hình 4.9 Thêm Gvr Reticle Pointer cho GvrReticalPointer

Gvr Reticle Pointer có chức năng đó là mỗi khi người chơi nhìn chằm vào một vật gì nó sẽ vẽ ra một hình tròn để người chơi sau đó có thể chú ý hơn vào nó (nó sẽ gọi hàm Render để render hình ảnh rõ nét hơn để người chơi có thể nhìn rõ hơn).

Để xử lý con trỏ trên màn hình VR của người chơi ta thêm 2 script sau:



Hình 4.10 Thêm Gvr Pointer Manager và Gvr Pointer Input Module vào pointerwalid.

4.3.3 Tạo độ dãn hình ảnh khi sử dụng VR

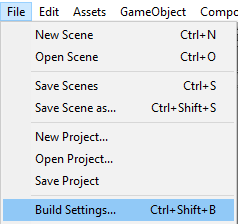
Muốn cho hình ảnh đẹp hơn ta dùng kỹ thuật render ảnh cho VR. Bằng cách thêm 2 Script: GvrPreRendering và GvrPostRendering.

Ở đây trò chơi của ta cũng không cần phải render cho ảnh nhằm tăng hiệu suất cho trò chơi.

4.4 Thiết lập và chạy ứng dụng

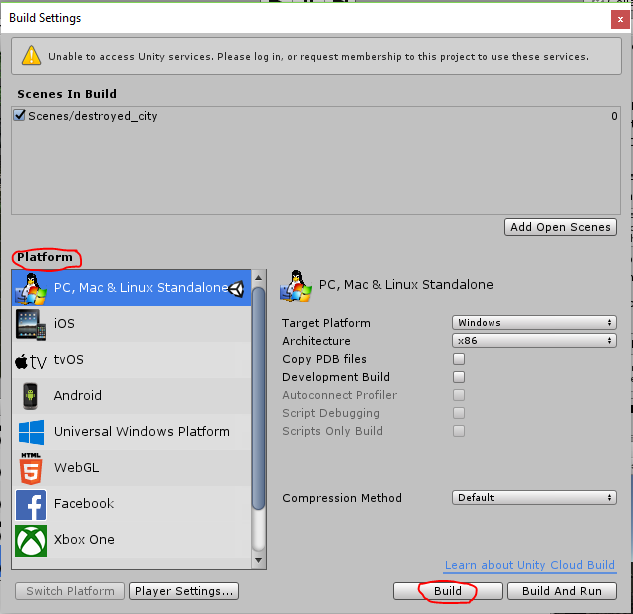
4.4.1 Xuất bản (Build) trò chơi

Để build trò chơi, ta vào File chọn Build Settings…



Hình 4.11 Mở chức năng Build của Unity

Ở mục Platform ta sẽ chọn thiết bị mà trò chơi sẽ chạy trên. Chọn xong ta bấm Build.



Hình 4.12 Chọn thiết bị mà trò chơi sẽ chạy trên nó

Đối với máy tính, khi build xong ta sẽ được trò chơi và ta có thể chơi ngay lập tức.

Tuy nhiên đối với Android, sau khi quá trình Build hoàn tất ta mới được file APK. Để trò chơi hoạt động trên điện thoại, ta sẽ dán file APK này vào điện thoại, cài đặt và cuối cùng là mới mở trò chơi. Hoặc ta có thể làm theo cách, cắm kết nối điện thoại vô máy tính, thì sau quá trình Build file APK sẽ có sẵn trên máy luôn không cần phải copy từ máy tính sang điện thoại.

Cuối cùng nếu hứng thú ta có thể chia sẻ trò chơi lên trang web Oculus để mọi người có thể tải về chơi.

4.4.2 Chạy trò chơi trên nền mạng LAN

Để mạng LAN có thể chạy trên mạng LAN, việc đầu tiên đó là ta cần phải có 2 máy chơi trên chung 1 mạng (wifi cùng 1 phòng, wifi do máy này phát máy kia thu,…). Tiếp tục, một người sẽ chọn máy mính làm host (server) để tổ chức trò chơi để các máy con (client) sau đó có thể tham gia.

4.5 Kết luận

Sau khi thực hiện xong chương này, ta đã có được một trò chơi hoàn chỉnh trên nền công nghệ VR. Cũng như ta cũng biết được cách xuất bản một trò chơi để chạy trên điện thoại. Ngoài ra trò chơi của ta còn có thể tổ chức cho 2 người chơi tham gia trên cùng một mạng LAN.

**TÀI LIỆU THAM KHẢO**

**Tiếng Anh**

1. <https://en.wikipedia.org/wiki/Virtual_reality>
2. <https://docs.unity3d.com/Manual/UNetOverview.html>
3. <https://docs.unity3d.com/Manual/UNetUsingHLAPI.html>
4. <https://docs.unity3d.com/Manual/UNetConcepts.html>

**PHỤ LỤC**