CHƯƠNG 1:TỔNG QUAN ĐỀ TÀI

1. Giới thiệu

Game bắn súng online góc nhìn thứ nhất là tựa game được viết bằng ngôn ngữ C#, bằng nền tảng công nghệ Unity 3D và kết hợp với Photon Unity Network (PUN). Game cho phép nhiều người chơi tham gia cùng lúc, chiến đấu với nhau bằng súng trong một khoản thời gian và người chiến thắng là người có số hạ gục người chơi khác nhiều nhất.

Game có các chức năng như tạo phòng chơi, tham gia phòng chơi của người khác, gây sát thương lẫn nhau bằng vũ khí.

Với mục đích cho người chơi được giải trí cùng lúc với nhau, tận hưởng khoảng thời gian giải trí lành mạnh, nhanh gọn và nhẹ không cầu kì nên Game bắn súng online góc nhìn thứ nhất (ColorGun) ra đời.

2. Khảo sát hiện trạng

Dòng game FPS là một dòng game đã có từ lâu và được ưa chuộng cho đến nay như các tựa game nổi tiếng như CS:GO, Call Of Duty,.. Với đặc điểm của dòng game này là dễ chơi, mức độ giải trí cao, làm cho người chơi hào hứng thì nói dòng game FPS là đang đứng đầu trong thị trường game cũng không ngoa.

Tuy là nhiều nhưng đa số các game hiện nay trên thị trường đều là các tựa game nặng, muốn chơi chúng ta phải tạo tài khoản, thủ tục trước khi chơi rất rườm rà cản trở người chơi, vì thế một ý tưởng tạo ra con game FPS chơi nhiều người cùng lúc, nhẹ, không cần phải lập tài khoản ra đời. ColorGun là thứ tôi đã nảy ra ý tưởng đó.

3. Đối tượng và mô tả đối tượng nghiên cứu

Các đối tượng cần nghiên cứu để có thể tạo ra một tựa game hoàn chỉnh bằng Unity.

+ Unity Engine 3D: Nghiên cứu về Pattern Design của Unity, Particle System, các package từ Asset Store, animation tạo hoạt ảnh.
+ Photon Unity Network: Nghiên cứu và tìm cách tích hợp vào Unity, xây dựng các chức năng online của game sử dụng Photon.
+ Ngôn ngữ C#: Nghiên cứu, tiếp thu cách tối ưu hóa các hàm, biến. Mạnh mẽ logic hơn làm game nhẹ hơn để có thể máy nào cũng chạy được.

4. Mục tiêu nghiên cứu

+ Ứng dụng kiến thức đã học và kinh nghiệm thực tế tại doanh nghiệp xây dựng, thiết kế một tựa game hoàn chỉnh bằng Unity Engine.
+ Nghiên cứu, sử dụng công nghệ Photon Unity Network vào tựa game.
+ Sử dụng Photon Unity Network để xây dựng server kết nối người chơi.
+ Sử dụng C# xây dựng các chức năng chính tạo thành một con game hoàn chỉnh.
+ Sử dụng các Asset có sẵn trong Unity Asset Store để xây dựng game.
+ Xuất game để có thể chạy trên windows và kết nối nhiều người chơi cùng lúc.

5. Phạm vi đề tài

Đề tài được nghiên cứu và xây dựng trong khoảng thời gian là ba tháng từ tháng 6/2023 đến tháng 9/2023**.**

Kiến thức nghiên cứu Unity, các package, animation, pattern design là từ kiến thức đã học và kinh nghiệp thực tập tại doanh nghiệp.

Kiến thức về ngôn ngữ C# là từ kiến thức đã học và kinh nghiệm rèn luyện, từ các diễn đàn trên internet.

Kiến thức về Photon Unity Network được tôi nghiên cứu từ tài liệu chính thức do Exit Game – Đơn vị tạo ra Photon Unity Network đưa ra.

6. Cấu trúc đề tài

Đề tài gồm hai phần:

- Bản báo cáo gồm các chương:
  + Chương 1: Tổng quan về đề tài.
  + Chương 2: Cơ sở lý thuyết.
  + Chương 3: Phân tích và thiết kế hệ thống games.
  + Chương 4: Thực nghiệm và triển khai.
  + Kết luận.
    - Kết quả đạt được.
    - Hướng phát triển.
  + Tài liệu tham khảo.

- Sản phẩm thực tế: File project Unity Game và bản build ra chạy trên hệ điều hành Windows.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

1. Ngôn ngữ sử dụng

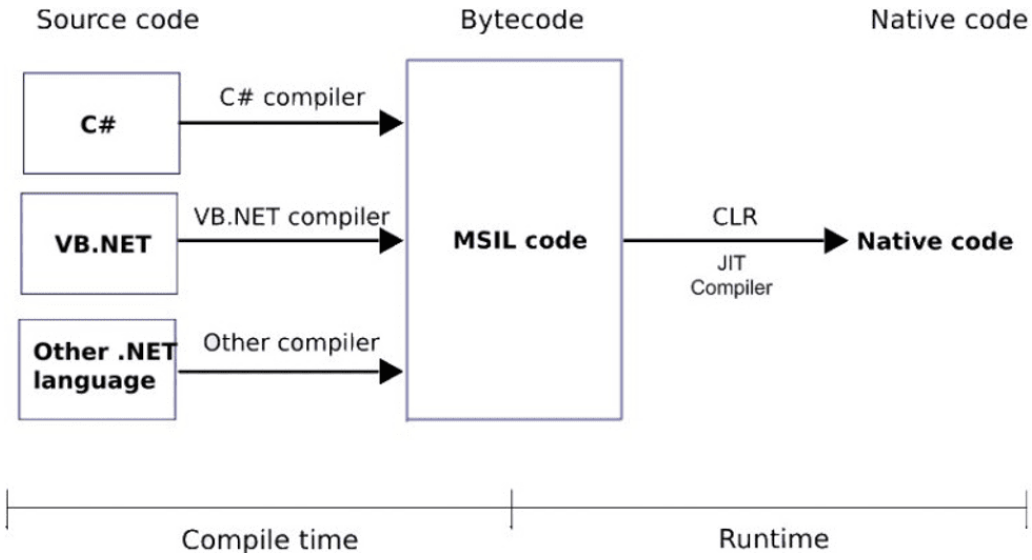
1.1 C sharp (C#)

C# (hay C sharp) là một ngôn ngữ lập trình đơn giản, được phát triển bởi đội ngũ kỹ sư của Microsoft vào năm 2000. C# là ngôn ngữ lập trình hiện đại, hướng đối tượng và được xây dựng trên nền tảng của hai ngôn ngữ mạnh nhất là C++ và Java.

Trong các ứng dụng Windows truyền thống, mã nguồn chương trình được biên dịch trực tiếp thành mã thực thi của hệ điều hành.

Trong các ứng dụng sử dụng .NET Framework, mã nguồn chương trình (C#, VB.NET) được biên dịch thành mã ngôn ngữ trung gian MSIL (Microsoft intermediate language).

Sau đó mã này được biên dịch bởi Common Language Runtime (CLR) để trở thành mã thực thi của hệ điều hành. Hình bên dưới thể hiện quá trình chuyển đổi MSIL code thành native code.



Hình 2.1 ***Quá trình chuyển đổi MSIL code thành native code.***

C# với sự hỗ trợ mạnh mẽ của .NET Framework giúp cho việc tạo một ứng dụng Windows Forms hay WPF (Windows Presentation Foundation), phát triển game, ứng dụng Web, ứng dụng Mobile trở nên rất dễ dàng.

1.2 Đặc trưng của ngôn ngữ C#

+ C# là ngôn ngữ đơn giản

C# loại bỏ một vài sự phức tạp và rối rắm của những ngôn ngữ như Java và c++, bao gồm việc loại bỏ những macro, những template, đa kế thừa, và lớp cơ sở ảo (virtual base class).

Ngôn ngữ C# đơn giản vì nó dựa trên nền tảng C và C++. Nếu chúng ta thân thiện với C và C++ hoặc thậm chí là Java, chúng ta sẽ thấy C# khá giống về diện mạo, cú pháp, biểu thức, toán tử và những chức năng khác được lấy trực tiếp từ ngôn ngữ C và C++, nhưng nó đã được cải tiến để làm cho ngôn ngữ đơn giản hơn.

+ C# là ngôn ngữ hiện đại

Điều gì làm cho một ngôn ngữ hiện đại? Những đặc tính như là xử lý ngoại lệ, thu gom bộ nhớ tự động, những kiểu dữ liệu mở rộng, và bảo mật mã nguồn là những đặc tính được mong đợi trong một ngôn ngữ hiện đại. C# chứa tất cả những đặc tính trên. Nếu là người mới học lập trình có thể chúng ta sẽ cảm thấy những đặc tính trên phức tạp và khó hiểu. Tuy nhiên, cũng đừng lo lắng chúng ta sẽ dần dần được tìm hiểu những đặc tính qua các nội dung khoá học này.

+ C# là một ngôn ngữ lập trình thuần hướng đối tượng

Lập trình hướng đối tượng (OOP: Object-oriented programming) là một phương pháp lập trình có 4 tính chất. Đó là tính trừu tượng (abstraction), tính đóng gói (encapsulation), tính đa hình (polymorphism) và tính kế thừa (inheritance). C# hỗ trợ cho chúng ta tất cả những đặc tính trên.

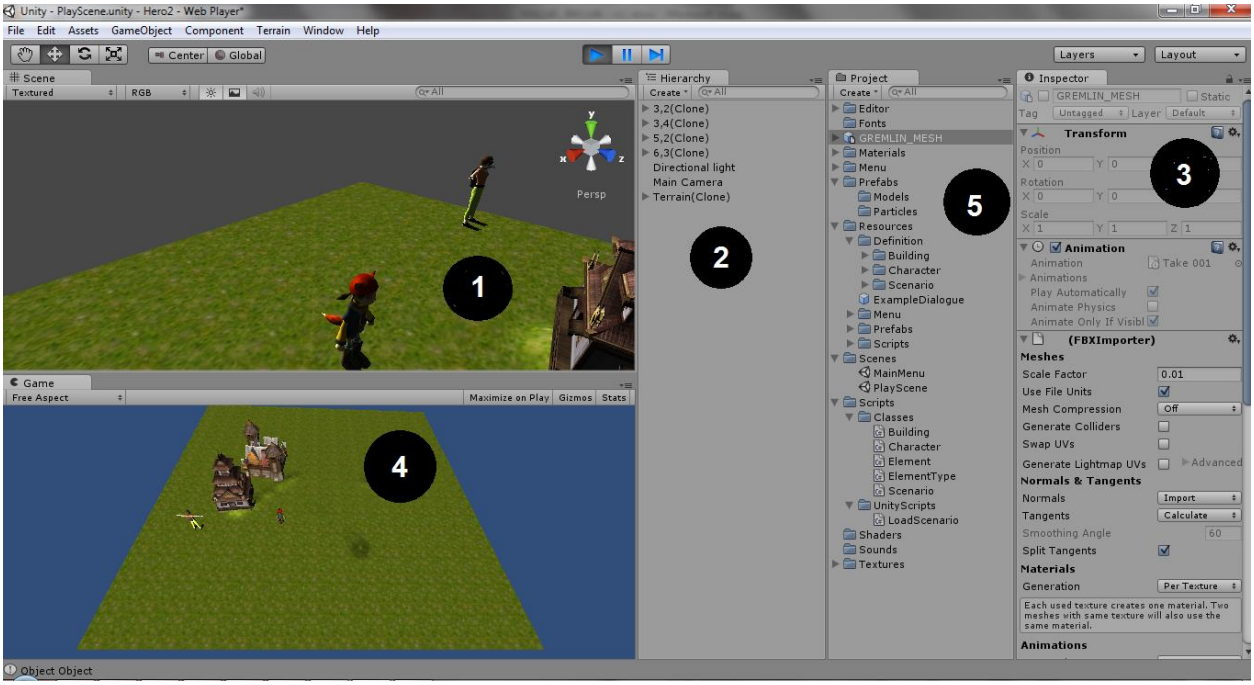
+ C# là một ngôn ngữ ít từ khóa

C# là ngôn ngữ sử dụng giới hạn những từ khóa. Phần lớn các từ khóa được sử dụng để mô tả thông tin. Chúng ta có thể nghĩ rằng một ngôn ngữ có nhiều từ khóa thì sẽ mạnh hơn. Điều này không phải sự thật, ít nhất là trong trường hợp ngôn ngữ C#, chúng ta có thể tìm thấy rằng ngôn ngữ này có thể được sử dụng để làm bất cứ nhiệm vụ nào.

2. Engine sử dụng

2.1 Unity Engine

Unity là một “cross- flatform game engine” tạm hiểu là công cụ phát triển game đa nền tảng được phát triển bởi Unity Technologies. Game engine này được sử dụng để phát trển game trên PC, Consoles, thiết bị di động và trên websites.



Hình 2.2 Giao diện trong Unity

Như hình trên chúng ta thấy có 5 khung khác nhau:

+ Scene [1] – nơi xây dựng trò chơi
+ Hierarchy [2] – danh sách các GameObject trong một cảnh game
+ Inspector [3] – màn hình cài đặt cho tài nguyên/đối tượng đang được chọn
+ Game [4] – cửa sổ xem trước game, chỉ hoạt động ở chế độ chơi (khi nhấn Play)
+ Project [5] – danh sách các tài nguyên trong project, đóng vai trò như một thư viện

[1].Cửa sổ Scene và Hierarchy

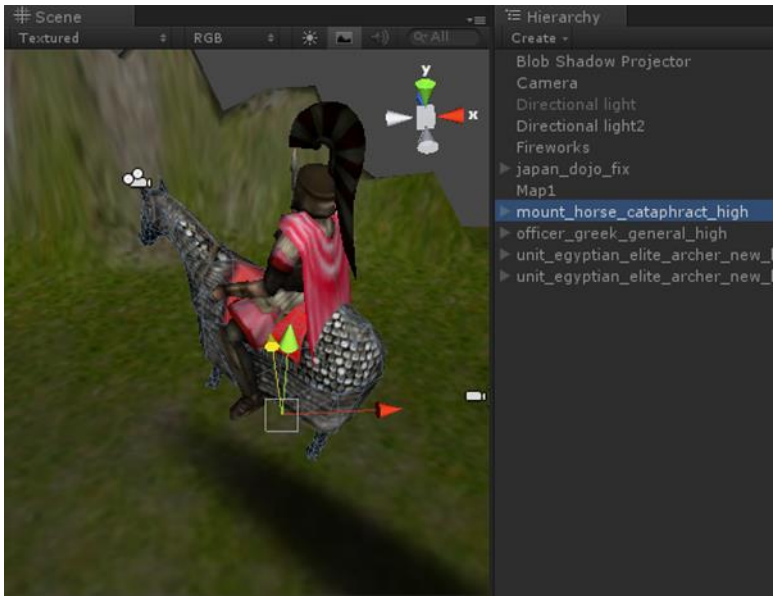
Cửa sổ Scene là nơi chúng ta sẽ xây dựng toàn bộ các đối tượng trong game. Cửa sổ cung cấp nhiều góc nhìn khác nhau, có thể nhìn dạng phối cảnh hoặc dạng song song. Chúng ta có thể kéo thả đối tượng trên cửa sổ này, di chuyển, xoay…

Hình 2.3 *Công cụ điều khiển màn hình Scene*

Cửa sổ Scene cũng kèm theo bốn nút điều khiển tiện lợi như hình trên. Truy cập từ bàn phím bằng cách sử dụng các phím Q, W, E, và R. Các phím thực hiện các hoạt động sau đây:

+ Công cụ bàn tay [Q]: công cụ này cho phép di chuyển trong cửa sổ Scene, xoay góc nhìn, phóng to, thu nhỏ góc nhìn.
+ Công cụ di chuyển [W]: Công cụ này dùng để di chuyển một đối tượng.
+ Công cụ xoay [E]: Công cụ này cho phép chúng ta xoay nhân vật theo một trục nào đó trong không gian.
+ Công cụ tỷ lệ [R]: Công cụ này cho phép chúng ta tăng giảm tỷ lệ kích thước của đối tượng.

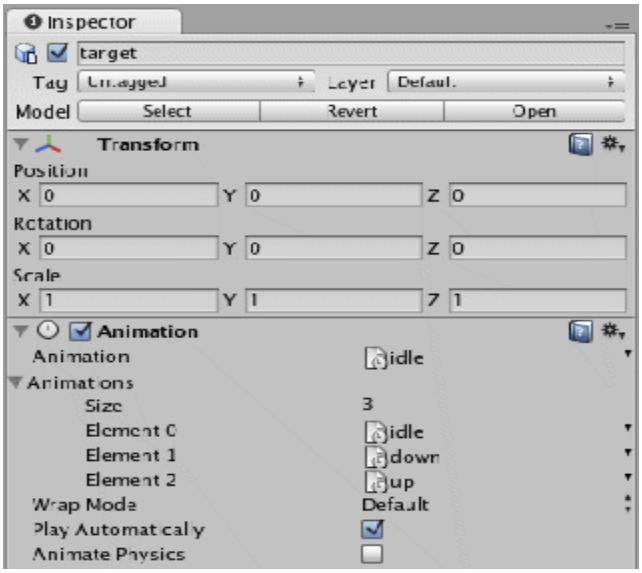
Khi chọn một đối tượng trong cửa sổ Scene, đối tượng này sẽ được tự động chọn trong cửa sổ Hierarchy và ngược lại.



Hình 2.4 ***Chọn đối tượng trong Scene và Hierrarchy***

[2].Inspector

Inspector sẽ hiển thị tất cả thông tin, các thành phần trong đối tượng game đang chọn, và cho phép điều chỉnh các biến của các thành phần này. Có thể xem cửa sổ này như cửa sổ Properties khi design giao diện Winform trên Visual Studio.



Hình 2.5 ***Cửa sổ Inspector hiển thị thông tin một đối tượng***

[3].Cửa sổ Game

Cửa sổ này sẽ hiển thị những gì có trong cửa sổ Scene và sẽ hoạt động khi nhấn nút Play. Trong cửa sổ này chúng ta có thể chọn các kích cỡ hiển thị khác nhau để build cho các loại máy khác nhau. Chúng ta có thể chơi thử game trên cửa sổ này khi đã nhấn nút Play. Lưu ý rằng khi cửa sổ này hoạt động rồi thì mọi chỉnh sửa trên cửa sổ Scene và cài đặt cho các đối tượng chỉ là tạm thời và khi nhấn nút Stop, cửa sổ này về lại trạng thái tĩnh thì mọi chỉnh sửa trước đó là không còn.

[4].Cửa sổ Project

Cửa sổ Project thể hiện nội dung bên trong thư mục Assets của project chúng ta. Khi thêm tài nguyên vào thư mục Assets ngay lập tức chúng sẽ tự động được cập nhập vào project Unity của chúng ta.

2.2 Định nghĩa lập trình trên Unity là gì ?

Lập trình Unity 2D và 3D được lập trình dựa vào 3 ngôn ngữ chính là C#, Boo và UnityScript. Thông thường, ngôn ngữ chính mà lập trình viên Unity sử dụng phổ biến nhất hiện nay là C#.

Đến đây, có câu hỏi đặt ra rằng: Tại sao Unity lại được sử dụng để thiết kế game rồi mà vẫn còn phải biết rõ những lập trình ở phía trên?

Unity có phần hạn chế lập trình nằm ở phần kéo thả nên các lập trình trên được áp dụng như phương pháp bổ trợ.

Như ví dụ sau: Một trò chơi được lập trình với thao tác đơn giản là bắn máy bay, ngoài công việc kéo thả các giao diện tại main menu, đặt nhân vật và khung nền,... thì nó còn được thực hiện dựa trên phương pháp drag and drop.

Những hiệu ứng bên trong trò chơi sẽ tự động lặp đi lặp lại như bầu trời của nền và đi lùi liên tục nhằm tạo ra các hiệu ứng máy bay đang di chuyển, vụ nổ, hoặc bắn đạn,... Chính vì vậy, không thể pick mãi 1 đối tượng rồi tiến hành thả liên tục trên đó cũng như cảnh chỉnh mãi bằng tay. Bởi đây là một trong những hiệu ứng tự động và cũng là cách duy nhất mà ta cần phải lập trình.

2.3 Các khái niệm quan trọng trong lập trình Unity ?

- GameObject

Một đối tượng cụ thể trong game gọi là một game object, có thể là nhân vật, đồ vật nào đó. Ví dụ: cây cối, xe cộ, nhà cửa, người...

- Component

Một GameObject sẽ có nhiều thành phần cấu tạo nên nó như là hình ảnh (sprite render), tập hợp các hành động (animator), thành phần xử lý va chạm (collision), tính toán vật lý (physical), mã điều khiển (script), các thành phần khác... mỗi thứ như vậy gọi là một component của GameObject.

- Sprite

Là một hình ảnh 2D của một game object có thể là hình ảnh đầy đủ, hoặc có thể là một bộ phận nào đó.

- Animation

Là tập một hình ảnh động dựa trên sự thay đổi liên tục của nhiều sprite khác nhau.

- Key Frame

Key Frame hay Frame là một trạng thái của một animation. Có thể được tạo nên từ 1 sprite hay nhiều sprite khác nhau.

- Prefabs

Là một khái niệm trong Unity, dùng để sử dụng lại các đối tượng giống nhau có trong game mà chỉ cần khởi tạo lại các giá trị vị trí, tỉ lệ biến dạng và góc quay từ môt đối tượng ban đầu. Ví dụ: Các đối tượng là đồng tiên trong game Mario đều có xử lý giống nhau,

nên ta chỉ việc tạo ra một đối tượng ban đầu, các đồng tiền còn lại sẽ sử dụng prefabs. Hoặc khi ta lát gạch cho một cái nền nhà, các viên gạch cũng được sử dụng là prefabs.

- Sounds

Âm thanh trong game.

- Script

Script là tập tin chứa các đoạn mã nguồn, dùng để khởi tạo và xử lý các đối tượng trong game. Trong Unity có thể dùng C#, Java Script, BOO để lập trình Script.

- Scenes

Quản lý tất cả các đối tượng trong một màn chơi của game.

- Assets

Bao gồm tất cả những gì phục vụ cho dự án game như sprite, animation, sound, script, scenes…

- Camera

Là một game object đặc biệt trong scene, dùng để xác định tầm nhìn, quansát các đối tượng khác trong game.

- Transform

Là 3 phép biến đổi tịnh tiến, quay theo các trục, và phóng to thu nhỏ một đối tượng

3. Photon Unity Network (PUN)

Photon Unity Networking (PUN) là gói Unity dành cho các trò chơi nhiều người chơi. Kết nối linh hoạt đưa người chơi các phòng nơi các đối tượng có thể được đồng bộ hóa qua mạng. RPC, Tùy chỉnh thuộc tính hoặc các sự kiện Photon "low level" là một trong số các tính năng cơ bản. Giao tiếp nhanh chóng và đáng tin được thực hiện thông qua (các) máy chủ Photon chuyên dụng, do đó người dùng không cần kết nối từng cái một.

Cũng giống như Unity Networking, Photon Unity Networking sử dụng UDP để liên lạc, nhưng điểm khác biệt chính là người chơi không trực tiếp lưu trữ máy chủ. Thay vào đó, khách hàng kết nối với một cụm máy chủ (được gọi là “đám mây”) và yêu cầu một phòng. Một phòng sẽ được tạo trên một trong các máy chủ trong cụm và khách hàng sẽ kết nối với phòng này.

Ngoài ra, thay vì có một “máy chủ” (thường là người chơi lưu trữ máy chủ), Photon Unity Networking định nghĩa một “máy khách chính”. Theo mặc định, người chơi tạo

phòng là khách hàng chính. Tuy nhiên, nếu người chơi đó rời đi, người chơi khác sẽ được chọn làm khách hàng chính. Ngược lại, điều này với Unity Networking, trong đó nếu máy chủ rời đi, trò chơi sẽ kết thúc hoặc phải được di chuyển, trong PUN, trò chơi có thể tiếp tục một cách liền mạch.

Hình 2.6 *Photon Unity Neworking*

3.1 Luồng sự kiện để sử dụng Photon Unity Networking cơ bản tạo một trò chơi online

+ Tạo một máy chủ:

Để tạo một máy chủ, chúng ta cần khởi tạo nó trên mạng và đăng ký nó với máy chủ chính. Việc khởi tạo yêu cầu số lượng người chơi tối đa và số cổng (25000). Đối với việc đăng ký máy chủ, tên của trò chơi phải là duy nhất, nếu không ta có thể gặp rắc rối với các dự án khác sử dụng cùng tên. Tên phòng có thể là bất kỳ tên nào.

```
[sourcecode language=”Csharp”]
private const string typeName = "UniqueGameName";
private const string gameName = "RoomName";

private void StartServer()
{
Network.InitializeServer(4, 25000, !Network.HavePublicAddress());
MasterServer.RegisterHost(typeName, gameName);
}
```

[/sourcecode]

Nếu máy chủ được khởi tạo thành công, OnServerInitialized() sẽ được gọi. Hiện tại, ta sẽ nhận được phản hồi cho biết máy chủ đã thực sự được khởi tạo.

[sourcecode language=”Csharp”]

void OnServerInitialized()

{

Debug.Log("Server Initializied");

}

[/sourcecode]

Tất cả những gì chúng ta cần bây giờ là một số dạng đầu vào để cho phép khởi động máy chủ khi chúng ta muốn. Để kiểm tra, ta tạo các nút bằng Unity GUI. Tôi chỉ muốn nhìn thấy các nút này nếu ta chưa khởi động máy chủ hoặc tham gia máy chủ, vì vậy nút này sẽ tự hiển thị nếu người dùng không phải là máy khách cũng như máy chủ.

[sourcecode language=”Csharp”]

void OnGUI()

{

if (!Network.isClient && !Network.isServer)

{

if (GUI.Button(new Rect(100, 100, 250, 100), "Start Server"))

StartServer();

}

}

[/sourcecode]

Bây giờ là lúc để kiểm tra những gì chúng ta đã làm cho đến hiện tại. Khi bắt đầu dự án, tất cả những gì ta thấy bây giờ là nút khởi động máy chủ. Nếu ta nhấn nút này, một thông báo sẽ hiển thị trong bảng điều khiển cho biết ta vừa khởi tạo máy chủ. Sau đó nút sẽ biến mất.

+ **Tham gia một máy chủ:**

Bây giờ chúng ta có chức năng tạo máy chủ nhưng chưa thể tìm kiếm các máy chủ hiện có hoặc tham gia một trong số chúng. Để làm được điều này, chúng ta cần gửi yêu cầu đến

máy chủ chính để lấy danh sách HostData. Nó chứa tất cả dữ liệu cần thiết để tham gia một máy chủ. Sau khi nhận được danh sách máy chủ, một thông báo sẽ được gửi đến OnMasterServerEvent().

Hàm này được gọi cho một số sự kiện, vì vậy chúng ta cần thêm một trình kiểm tra để xem liệu thông báo có bằng MasterServerEvent.HostListReceived hay không . Nếu đúng như vậy, chúng ta có thể lưu trữ danh sách máy chủ.

[sourcecode language="Csharp"]

```
private HostData[] hostList;

private void RefreshHostList()
{
MasterServer.RequestHostList(typeName);
}

void OnMasterServerEvent(MasterServerEvent msEvent)
{
if (msEvent == MasterServerEvent.HostListReceived)
hostList = MasterServer.PollHostList();
}
```

[/sourcecode]

Để tham gia một máy chủ, tất cả những gì chúng ta cần là một mục trong danh sách máy chủ. OnConnectedToServer () được gọi sau khi chúng ta tham gia vào máy chủ. Chúng ta sẽ mở rộng chức năng này sau

[sourcecode language="Csharp"]

```
private void JoinServer(HostData hostData)
{
Network.Connect(hostData);
}

void OnConnectedToServer()
```

```
{
Debug.Log("Server Joined");
}
[/sourcecode]
```

Bằng cách mở rộng GUI với một số nút bổ sung, các chức năng chúng ta vừa tạo có thể được gọi. Lúc này sẽ có hai nút, một nút để khởi động máy chủ và một nút khác để làm mới danh sách máy chủ. Một nút mới được tạo cho mỗi máy chủ và nó sẽ kết nối người dùng với phòng tương ứng.

```
[sourcecode language=”Csharp”]
void OnGUI()
{
if (!Network.isClient && !Network.isServer)
{
if (GUI.Button(new Rect(100, 100, 250, 100), "Start Server"))
StartServer();

if (GUI.Button(new Rect(100, 250, 250, 100), "Refresh Hosts"))
RefreshHostList();

if (hostList != null)
{
for (int i = 0; i < hostList.Length; i++)
{
if (GUI.Button(new Rect(400, 100 + (110 * i), 300, 100), hostList[i].gameName))
JoinServer(hostList[i]);
}
}
}
[/sourcecode]
```

Bằng cách trên, chúng ta đã tạo máy chủ và tham gia vào máy chủ đó thành công, việc còn lại để trở thành một con game hoàn chỉnh là tạo ra các người chơi tượng trưng cho người tham gia, đồng bộ trạng thái – cử động, … để thành một con game hoàn chỉnh.

+ **Tạo người chơi:**

Bây giờ chúng ta có thể kết nối nhiều người chơi với nhau, giờ đây chúng ta có thể mở rộng cơ chế trò chơi. Thiết lập một khung cảnh đơn giản với mặt Plane, camera và Lighting system. Thêm một game object cho người chơi và cố định các chuyển động quay để ta không có hành vi lạ khi di chuyển.

```
[sourcecode language=”Csharp”]
public class Player : MonoBehaviour
{
public float speed = 10f;

void Update()
{
InputMovement();
}

void InputMovement()
{
if (Input.GetKey(KeyCode.W))
rigidbody.MovePosition(rigidbody.position + Vector3.forward * speed *
Time.deltaTime);

if (Input.GetKey(KeyCode.S))
rigidbody.MovePosition(rigidbody.position – Vector3.forward * speed *
Time.deltaTime);

if (Input.GetKey(KeyCode.D))
```

rigidbody.MovePosition(rigidbody.position + Vector3.right * speed * Time.deltaTime);

if (Input.GetKey(KeyCode.A))

rigidbody.MovePosition(rigidbody.position – Vector3.right * speed * Time.deltaTime);

}

}

[/sourcecode]

Tiếp theo, thêm thành phần network view component vào đối tượng (Component > Miscellaneous > Network View). Điều này sẽ cho phép ta gửi các gói dữ liệu qua mạng để đồng bộ hóa đối tượng. Trường đồng bộ hóa trạng thái được tự động đặt thành "reliable delta compressed". Điều này có nghĩa là dữ liệu đã đồng bộ hóa sẽ được gửi tự động nhưng chỉ khi giá trị của nó thay đổi. Vì vậy, ví dụ nếu ta di chuyển với tư cách là người chơi, vị trí của ta sẽ được cập nhật trên máy chủ. Bằng cách đặt nó thành "off", sẽ không có đồng bộ hóa tự động nào cả và ta phải thực hiện thủ công. Bây giờ hãy đặt nó thành "reliable".

Trong tập lệnh NetworkManager, hãy thêm một biến đối tượng trò chơi công khai cho prefab đối tượng. Trong hàm mới SpawnPlayer() , prefab sẽ được khởi tạo trên mạng, vì vậy tất cả khách hàng sẽ thấy đối tượng này trong trò chơi của họ. Nó đòi hỏi một vị trí, sự xoay vòng và nhóm, vì vậy tôi khuyên ta nên tạo một điểm hồi sinh.

[sourcecode language="Csharp"]

public GameObject playerPrefab;

void OnServerInitialized()

{

SpawnPlayer();

}

void OnConnectedToServer()

{

SpawnPlayer();

}

private void SpawnPlayer()

{

Network.Instantiate(playerPrefab, new Vector3(0f, 5f, 0f), Quaternion.identity, 0);

}

[/sourcecode]

Tạo bản dựng mới và chạy lại hai phiên bản. Ta sẽ nhận thấy rằng ta có thể kiểm soát tất cả người chơi được kết nối, không chỉ của riêng ta. Điều này cho thấy một khía cạnh quan trọng sẽ được sử dụng thường xuyên khi tạo trò chơi nhiều người chơi: ai điều khiển đối tượng nào?

Một cách để khắc phục sự cố này là xây dựng một quy trình kiểm tra mã trình phát để nó chỉ nhận đầu vào từ người dùng đã khởi tạo đối tượng. Vì ta đặt đồng bộ hóa đáng tin cậy trên chế độ xem mạng nên dữ liệu sẽ được gửi tự động qua mạng và không cần thông tin nào khác.

Để triển khai phương pháp này, chúng ta cần kiểm tra xem đối tượng trên mạng có phải là của tôi hay không trong tập lệnh trình phát. Thêm câu lệnh if sau vào Update()

[sourcecode language=”Csharp”]

void Update()

{

if (networkView.isMine)

{

InputMovement();

}

}

[/sourcecode]

Nếu bây giờ ta thực hiện một bài kiểm tra khác, ta sẽ thấy mình chỉ có thể điều khiển một trong những người chơi.

Một giải pháp khác có thể là gửi tất cả thông tin đầu vào đến máy chủ, sau đó máy chủ sẽ chuyển đổi dữ liệu của ta thành chuyển động thực tế và gửi lại vị trí mới của ta cho mọi người trên mạng. Ưu điểm là mọi thứ đều được đồng bộ hóa trên máy chủ. Điều này ngăn cản người chơi gian lận với khách hàng địa phương của họ. Nhược điểm của phương pháp này là độ trễ giữa máy khách và máy chủ, điều này có thể dẫn đến việc người dùng phải chờ xem hành động mình thực hiện.

+ **Đồng bộ hóa trạng thái:**

Có hai phương pháp truyền thông mạng. Đầu tiên là Đồng bộ hóa trạng thái và cái còn lại là Cuộc gọi thủ tục từ xa, sẽ được đề cập trong một đoạn khác. Đồng bộ hóa trạng thái liên tục cập nhật các giá trị qua mạng. Cách tiếp cận này hữu ích với những dữ liệu thường xuyên thay đổi, chẳng hạn như chuyển động của người chơi. Trong hàm OnSerializeNetworkView(), các biến được gửi hoặc nhận và sẽ đồng bộ hóa chúng nhanh chóng và đơn giản. Để cho ta thấy cách thức hoạt động của nó, ta sẽ viết mã đồng bộ hóa vị trí của người chơi.

Chuyển đến thành phần xem mạng trên prefab của trình phát. Trường được quan sát chứa thành phần sẽ được đồng bộ hóa. Biến đổi được tự động thêm vào trường này, dẫn đến vị trí, góc quay và tỷ lệ được cập nhật tùy thuộc vào tốc độ gửi. Kéo thành phần của tập lệnh trình phát vào trường được quan sát để chúng ta có thể viết phương thức đồng bộ hóa của riêng mình.

Thêm OnSerializeNetworkView() vào tập lệnh trình phát. Hàm này được gọi tự động mỗi khi nó gửi hoặc nhận dữ liệu. Nếu người dùng đang ghi vào luồng, điều đó có nghĩa là anh ta đang gửi dữ liệu. Bằng cách sử dụng stream.Serialize() , biến sẽ được tuần tự hóa và được các máy khách khác nhận. Nếu người dùng nhận được dữ liệu, chức năng tuần tự hóa tương tự sẽ được gọi và hiện có thể được đặt để lưu trữ dữ liệu cục bộ. Lưu ý rằng thứ tự của các biến phải giống nhau khi gửi và nhận dữ liệu, nếu không các giá trị sẽ bị xáo trộn.

```
[sourcecode language="Csharp"]
void OnSerializeNetworkView(BitStream stream, NetworkMessageInfo info)
{
Vector3 syncPosition = Vector3.zero;
if (stream.isWriting)
```

{

syncPosition = rigidbody.position;

stream.Serialize(ref syncPosition);

}

else

{

stream.Serialize(ref syncPosition);

rigidbody.position = syncPosition;

}

}

[/sourcecode]

Tạo một bản Build khác và chạy nó. Kết quả sẽ giống như trước đây, nhưng bây giờ ta đã cấp cho mình quyền kiểm soát chuyển động và cách thức hoạt động của đồng bộ hóa.

+ **Thiết lập các giá trị:**

Ta có thể nhận thấy vấn đề về độ trễ giữa hai phiên bản do tốc độ gửi. Cài đặt tiêu chuẩn trong Unity là một gói đang được cố gắng gửi 15 lần mỗi giây. Vì mục đích thử nghiệm, ta sẽ thay đổi tốc độ gửi. Để thực hiện việc này, trước tiên hãy chuyển đến cài đặt mạng tại (Edit> Project Settings > Network). Sau đó, thay đổi tốc độ gửi thành 5, kết quả là sẽ gửi ít gói dữ liệu hơn. Nếu ta thực hiện một bản dựng và thử nghiệm khác, độ trễ sẽ rõ ràng hơn.

Để làm trơn tru quá trình chuyển đổi từ giá trị dữ liệu cũ sang giá trị dữ liệu mới và khắc phục các vấn đề về độ trễ này, có thể sử dụng phép nội suy. Có một số lựa chọn về cách thực hiện điều này. Đối với hướng dẫn này, ta sẽ nội suy giữa vị trí hiện tại và vị trí mới nhận được sau khi đồng bộ hóa.

OnSerializeNetworkView() cần được mở rộng để lưu trữ tất cả dữ liệu cần thiết: vị trí hiện tại, vị trí mới và độ trễ giữa các lần cập nhật.

[sourcecode language=”Csharp”]

private float lastSynchronizationTime = 0f;

private float syncDelay = 0f;

private float syncTime = 0f;

private Vector3 syncStartPosition = Vector3.zero;

```
private Vector3 syncEndPosition = Vector3.zero;

void OnSerializeNetworkView(BitStream stream, NetworkMessageInfo info)
{
Vector3 syncPosition = Vector3.zero;
if (stream.isWriting)
{
syncPosition = rigidbody.position;
stream.Serialize(ref syncPosition);
}
else
{
stream.Serialize(ref syncPosition);

syncTime = 0f;
syncDelay = Time.time – lastSynchronizationTime;
lastSynchronizationTime = Time.time;

syncStartPosition = rigidbody.position;
syncEndPosition = syncPosition;
}
}
[/sourcecode]
```

Ta đã kiểm tra Update() xem đối tượng có được người chơi điều khiển hay không. Ta cần thêm chức năng để khi không gặp trường hợp này, ta sẽ sử dụng phép nội suy giữa các giá trị được đồng bộ hóa.

```
[sourcecode language=”Csharp”]
void Update()
{
if (networkView.isMine)
```

{

InputMovement();

}

else

{

SyncedMovement();

}

}

private void SyncedMovement()

{

syncTime += Time.deltaTime;

rigidbody.position = Vector3.Lerp(syncStartPosition, syncEndPosition, syncTime / syncDelay);

}

[/sourcecode]

Tạo bản Build mới và kiểm tra nó, bây giờ ta sẽ thấy quá trình chuyển đổi trông đẹp hơn giữa các bản cập nhật.

+ **Prediction:**

Mặc dù quá trình chuyển đổi trông mượt mà nhưng ta nhận thấy có độ trễ nhỏ giữa đầu vào và chuyển động thực tế. Điều này là do vị trí được cập nhật sau khi nhận được dữ liệu mới. Cho đến khi chúng ta phát minh ra du hành thời gian, tất cả những gì chúng ta có thể làm là dự đoán điều gì sẽ xảy ra dựa trên dữ liệu cũ.

Một phương pháp để dự đoán vị trí tiếp theo là tính đến vận tốc. Vị trí cuối chính xác hơn có thể được tính bằng cách cộng vận tốc nhân với độ trễ.

[sourcecode language="Csharp"]

void OnSerializeNetworkView(BitStream stream, NetworkMessageInfo info)

{

Vector3 syncPosition = Vector3.zero;

Vector3 syncVelocity = Vector3.zero;

```
if (stream.isWriting)
{
syncPosition = rigidbody.position;
stream.Serialize(ref syncPosition);

syncVelocity = rigidbody.velocity;
stream.Serialize(ref syncVelocity);
}
else
{
stream.Serialize(ref syncPosition);
stream.Serialize(ref syncVelocity);

syncTime = 0f;
syncDelay = Time.time – lastSynchronizationTime;
lastSynchronizationTime = Time.time;

syncEndPosition = syncPosition + syncVelocity * syncDelay;
syncStartPosition = rigidbody.position;
}
}
[/sourcecode]
```

Sau khi Build và thử nghiệm lại trò chơi, ta sẽ nhận thấy quá trình chuyển đổi vẫn diễn ra suôn sẻ và độ trễ giữa đầu vào của ta và chuyển động thực tế dường như ít hơn. Ngoài ra còn có một số trường hợp đặc biệt mà hành vi có vẻ hơi lạ nếu độ trễ quá cao. Nếu người chơi bắt đầu di chuyển, những khách hàng khác vẫn dự đoán ta sẽ đứng yên. Đặt tốc độ gửi ở cài đặt mạng trở lại 15 bản cập nhật mỗi giây để có kết quả tốt hơn.

Ta sẽ sử dụng lưới điều hướng để di chuyển xung quanh và điều này dường như giúp cho việc nội suy và dự đoán tốt hơn. Tốc độ gửi được đặt thành 5 và với tư cách là người dùng, ta hầu như không thể nhận thấy độ trễ. Hướng dẫn này sẽ không đi sâu hơn vào việc

tích hợp navmesh, nhưng đối với các dự án tương lai của ta, ta có thể nên xem xét phương pháp này.

+ **Remote Procedure Calls:**

Một phương thức giao tiếp mạng khác là Cuộc gọi thủ tục từ xa (RPC), phương thức này hữu ích hơn đối với dữ liệu không thay đổi liên tục. Một ví dụ điển hình mà ta đã sử dụng những thứ này trong bản demo Kickstarter của mình là hộp thoại. Trong đoạn này, ta sẽ thay đổi màu sắc của trình phát qua mạng.

Những gì RPC thực hiện là gọi hàm trên thành phần xem mạng và thành phần này tìm kiếm chức năng RPC chính xác. Bằng cách thêm [RPC] vào trước hàm, nó có thể được gọi qua mạng. Cách tiếp cận này chỉ có thể gửi số nguyên, số float, chuỗi, networkViewID, vectơ và quaternions. Vì vậy không phải tất cả các tham số đều có thể được gửi đi, nhưng điều này có thể giải quyết được. Để gửi một đối tượng trò chơi, chúng ta nên thêm thành phần chế độ xem mạng vào đối tượng này để có thể sử dụng networkViewID của nó. Để gửi một màu, chúng ta nên chuyển đổi nó thành một vectơ hoặc quaternion.

Một RPC được gửi bằng cách gọi networkView.RPC() , trong đó ta xác định tên hàm và các tham số. Ngoài ra, chế độ RPC cũng được yêu cầu: “Máy chủ” chỉ gửi dữ liệu đến máy chủ, “Khác” cho mọi người trên máy chủ ngoại trừ chính ta và “Tất cả” gửi dữ liệu đó cho mọi người. Hai cái cuối cùng cũng có chức năng thiết lập là được lưu vào bộ đệm, điều này dẫn đến việc những người chơi mới kết nối sẽ nhận được tất cả các giá trị được lưu vào bộ đệm này. Bởi vì bây giờ ta gửi gói dữ liệu này đến mọi khung hình nên không cần phải đệm nó.

Để tích hợp chức năng này vào trò chơi hướng dẫn của ta, Update() cần gọi hàm này để kiểm tra đầu vào và thay đổi vật liệu thành màu ngẫu nhiên. Chức năng RPC thay đổi màu sắc dựa trên đầu vào và nếu đối tượng trình phát được người dùng điều khiển, anh ta sẽ gửi RPC đến tất cả những người khác trên mạng.

```
[sourcecode language=”Csharp”]
void Update()
{
if (networkView.isMine)
{
```

```
InputMovement();
InputColorChange();
}
else
{
SyncedMovement();
}
}

private void InputColorChange()
{
if (Input.GetKeyDown(KeyCode.R))
ChangeColorTo(new Vector3(Random.Range(0f, 1f), Random.Range(0f, 1f),
Random.Range(0f, 1f)));
}

[RPC] void ChangeColorTo(Vector3 color)
{
renderer.material.color = new Color(color.x, color.y, color.z, 1f);

if (networkView.isMine)
networkView.RPC("ChangeColorTo", RPCMode.OthersBuffered, color);
}
[/sourcecode]
```

Bây giờ hãy tạo một bản build khác và nếu ta chạy trò chơi, ta sẽ thấy màu sắc của trình phát có thể được thay đổi.

CHƯƠNG 3: PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG GAME

1. Các đối tượng, các thành phần trong game.

1.1 Các đối tượng trong Game.

- **Nhân vật (Player)**

- Nhân vật là một phần không thể thiếu trong một tựa game, không chỉ riêng tựa game bắn súng góc nhìn thứ nhất mà tôi đang thực hiện này.
- Ở tựa game này, nhân vật là đối tượng quan trọng nhất, có chức năng di chuyển như chạy, nhảy, quay 360 độ, đi qua 4 hướng trên, dưới, trái, phải.
- Nhân vật cũng là thứ mô phỏng, là thứ để cho người chơi có thể điều khiển để cầm vũ khí để bắn một nhân vật khác hay nhận sát thương từ nhân vật khác.
- Nhân vật trong trò chơi này đều có thanh sinh lực (Health), nếu thanh này hết do nhận sát thương từ Player khác quá nhiều thì nhân vật sẽ bị hạ gục.
- Nhân vật có thể tăng sinh lực bằng cách ăn ô vật phẩm hoặc dễ hơn là bị hạ gục và hồi sinh sẽ hồi lại đầy thanh
- Nhân vật có thể cầm súng, có thể hồi sinh ở bất cứ đâu trên bản đồ.

- **Bản đồ (Map)**

- Bản đồ là thứ không thể thiếu trong một tựa game không riêng gì dòng game FPS, nó mô phỏng lại môi trường sống, giúp cho các nhân vật có thể di chuyển, có thế đứng và có thể đặt các chướng ngại vật tăng độ khó cho game.
- Bản đồ trong tựa game này giúp nhân vật đứng, đi chuyển để hạ gục nhau, là nơi để có thể tránh. Nó bao gồm nhiều thứ gộp lại, dựng nên như: Tường (sphere), thùng (Cube), nền đất (Plane), …

- **Vũ khí súng (Gun)**

- Vũ khí là thứ không thể thiếu trong tựa game bắn súng FPS
- Trong tựa game này, vũ khí súng có chức năng bắn các tia (Raycast) vào các Player. Nếu trúng sẽ bị trừ sinh lực Player đối phương.
- Trong tựa game này, vũ khí súng còn có giới hạn được gọi là đạn, nếu bắn quá số lượng đạn thì sẽ không được bắn nữa, phải nạp đạn lại thì mới có thể bắn tiếp.

1.2 Các thành phần trong Game.

- **Màn hình Loading:** Màn hình loading là một thành phần quan trọng, trong game bắn súng mà tôi đang thực hiện, nó sẽ thể hiện quá trình đang tải, lấy dự liệu lên, đưa dữ liệu về từ server, tải màn chơi mới.
- **Màn hình Menu:** Màn hình menu cho phép người chơi lựa chọn các chức năng mong muốn như tạo phòng chơi, tham gia một phòng chơi có sẵn và bắt đầu chơi. Nó là thành phần quan trọng không thể thiếu.
- **Màn hình game:** Nó là màn hình diễn ra game, là màn hình chứa bản đồ, nhân vật và các đối tượng khác cùng nhau chiến đấu đến khi thắng cuộc.
- **Camera:** Là thành phần không thể thiếu, nó là góc nhìn để người chơi quan sát nhân vật của mình hiện tại như thế nào, các chỉ số của bản than, đã đi tới đâu và cầm vũ khí gì.

2. Các chiến lược chơi game

- Bởi vì game thuộc dòng bắn súng online FPS góc nhìn thứ nhất nên game chỉ có một vòng, có tổng cộng là một bản đồ để người chơi có thể chiến đấu với nhau.
- Game online nhiều người chơi với nhau nên độ khó là dựa vào thực lực của đối thủ, đối thủ có kỹ năng tốt thì coi như ván đấu đó khó.
- Mỗi vũ khí của người chơi đều có một số lượng đạn, bắn hết thì sẽ không bắn được nữa, tạo độ khó cho game.
- Người chơi có thể ăn máu để tăng sinh lực, hạn chế bị hạ gục.
- Game có cách tính điểm rất đơn giản, người chơi hạ gục người khác sẽ được tính là 1 kill, người chơi bị người khác hạ gục thì sẽ tính là 1 death. Ai hạ gục người khác nhiều kill nhất sẽ tính là chiến thắng.
- Mọi chỉ số như kill, death đều được lưu trữ vào biến trên Unity và được gửi lên Server. Các chỉ số đó sẽ được thể hiện ở bản kết quả trong ván đấu của game.

3. Các thuật toán áp dụng cho game

+ Thuật toán kết nối Server.

```
void Start()
{
 PhotonNetwork.Disconnect();
 Debug.Log("Connecting to Master");
 PhotonNetwork.ConnectUsingSettings();
}
3 references
public override void OnConnectedToMaster()
{
    Debug.Log("Connected to Master");
    PhotonNetwork.JoinLobby();
    PhotonNetwork.AutomaticallySyncScene = true;
}
3 references
public override void OnJoinedLobby()
{
    LobbyManager.Instance.OpenLobby("lobby");
    Debug.Log("Joined Lobby");
}
```

Hình 3.1 Thuật toán kết nối Server

- Trước tiên, khi vừa mới vào game, hàm Start() sẽ chạy đầu tiên, trước hết nó sẽ ngắt kết nối Disconnect() tất cả các phiên kết nối trước và sẽ kết nối bằng **PhotonNetwork.ConnectUsingSettings()**

+ Thuật toán tạo phòng, tham gia phòng và thoát phòng.

```
public override void OnJoinedRoom()
{
    LobbyManager.Instance.OpenLobby("room");
    roomNameText.text = PhotonNetwork.CurrentRoom.Name;

    Player[] players = PhotonNetwork.PlayerList;

    foreach(Transform child in playerListContent)
    {
        Destroy(child.gameObject);
    }
    for (int i = 0; i < players.Count(); i++)
    {
        Instantiate(playerListItemPrefab, playerListContent).GetComponent<PlayerItem>().SetUp(players[i]);

    }

    startGameBtn.SetActive(PhotonNetwork.IsMasterClient);
}

3 references
public override void OnMasterClientSwitched(Player newMasterClient)
{
    startGameBtn.SetActive(PhotonNetwork.IsMasterClient);
}
3 references
public override void OnCreateRoomFailed(short returnCode, string message)
{
    errorText.text = "Create Room Fail: " + message;
    LobbyManager.Instance.OpenLobby("error");
}
0 references
public void LeaveRoom()
{
    PhotonNetwork.LeaveRoom();
    LobbyManager.Instance.OpenLobby("loading");
}

1 reference
public void JoinRoom(RoomInfo info)
{
    PhotonNetwork.JoinRoom(info.Name);
    LobbyManager.Instance.OpenLobby("loading");
}
3 references
    0 references
    public void CreateRoom()
    {
        if(string.IsNullOrEmpty(roomNameInputField.text))
        {
            return;
        }
        PhotonNetwork.CreateRoom(roomNameInputField.text);
        LobbyManager.Instance.OpenLobby("loading");
    }
```

Hình 3.2 Tạo phòng, tham gia phòng và thoát phòng

- Khi đã kết nối Server xong, ta sẽ tạo phòng bằng hàm CreateRoom(), tham gia phòng bằng hàm JoinRoom() và thoát phòng bằng LeaveRoom()

- Các hàm OnJoinedRoom(), On… là các hàm thực thi.

- Hàm OnMasterClientSwitched() dùng để chuyển đổi chủ phòng khi người tạo phòng thoát khỏi phòng mà còn các người chơi khác trong phòng tránh bị xóa phòng.

+ Thuật toán tham gia trò chơi.

```
0 references
public void StartGame()
{
    PhotonNetwork.LoadLevel(2);
}
5 references
public override void OnRoomListUpdate(List<RoomInfo> roomList)
{
    foreach(Transform trans in roomListContent)
    {
        Destroy(trans.gameObject);
    }
    for (int i = 0; i < roomList.Count; i++)
    {
        if (roomList[i].RemovedFromList)
            continue;
        Instantiate(roomListItemPrefab, roomListContent).GetComponent<RoomItem>().SetUp(roomList[i]);
    }
}

3 references
public override void OnPlayerEnteredRoom(Player newPlayer)
{
    Instantiate(playerListItemPrefab, playerListContent).GetComponent<PlayerItem>().SetUp(newPlayer);
}
```

Hình 3.3 Thuật toán tham gia trò chơi

- Hàm StartGame() là hàm giúp cho các người chơi trong phòng có thể vào game

- Hàm OnRoomListUpdate() là hàm cập nhật danh sách người chơi, nếu người chơi có thoát game hay vào them.

- Hàm OnPlayerEnterRoom() cho phép tất cả người chơi vào phòng hay một người khác vào phòng khi phòng đã chơi.

+ Thuật toán triệu hồi nhân vật.

```
void Start()
{
    if (PV.IsMine)
    {
        CreateController();
    }
}
2 references
void CreateController()
{
    Transform spawnpoint = SpawnManager.Instance.GetSpawnpoint();
    controller = PhotonNetwork.Instantiate(Path.Combine("PhotonPrefabs", "PlayerController"), spawnpoint.position, spawnpoint.rotation, 0, new object[] { PV.ViewID });
}
```

Hình 3.4 Thuật toán triệu hồi nhân vật

- Hàm triệu hồi nhân vật thực chất là sử dụng Instantiate của unity, đưa đối tượng prefab vào game qua các vị trí đã chỉ định

+ Thuật toán di chuyển nhân vật, quay xung quanh và nhảy.

```
void Look()
{
    transform.Rotate(Vector3.up * Input.GetAxisRaw("Mouse X") * mouseSensitivity);

    verticalLookRotation += Input.GetAxisRaw("Mouse Y") * mouseSensitivity;
    verticalLookRotation = Mathf.Clamp(verticalLookRotation, -90f, 90f);

    cameraHolder.transform.localEulerAngles = Vector3.left * verticalLookRotation;
}

1 reference
void Move()
{
    Vector3 moveDir = new Vector3(Input.GetAxisRaw("Horizontal"), 0, Input.GetAxisRaw("Vertical")).normalized;

    moveAmount = Vector3.SmoothDamp(moveAmount, moveDir * (Input.GetKey(KeyCode.LeftShift) ? sprintSpeed : walkSpeed), ref smoothMoveVelocity, smoothTime);
}
 1 reference
 void Jump()
{
    if (Input.GetKeyDown(KeyCode.Space) && grounded)
    {
        rb.AddForce(transform.up * jumpForce);
    }
}
```

Hình 3.5 Thuật toán di chuyển nhân vật

+ Thuật toán trang bị súng.

```
void EquipItem(int _index)
{
    if (_index == previousItemIndex)
        return;
    itemIndex = _index;
    items[itemIndex].itemGameObject.SetActive(true);

    if (previousItemIndex != -1)
    {
        items[previousItemIndex].itemGameObject.SetActive(false);
    }

    previousItemIndex = itemIndex;

    if(PV.IsMine)
    {
        Hashtable hash = new Hashtable();
        hash.Add("itemIndex", itemIndex);
        PhotonNetwork.LocalPlayer.SetCustomProperties(hash);
    }
```

Hình 3.6 Thuật toán trang bị súng

- Hàm này cho phép đổi súng, trang bị vào nhân vật, sẽ có một list vũ khí, và khi người chơi muốn đổi súng sẽ quay con lăn chuột để đổi

- Hàm này được gắn vào một hàm lấy sự kiện con lăn chuột để thực thi.

+ Thuật toán bắn súng, gây sát thương.

```
void Shoot()
{
    if (ammo != 0 && reloadAni.isPlaying == false)
    {
        recoiling = true;
        recovering = false;
        Ray ray = cam.ViewportPointToRay(new Vector3(0.5f, 0.5f));
        ray.origin = cam.transform.position;
        ammo--;
        if (Physics.Raycast(ray, out RaycastHit hit))
        {
            hit.collider.gameObject.GetComponent<IDamageable>()?.TakeDamage(((gunInfo)itemInfo).damage);
            PV.RPC("RPC_Shoot", RpcTarget.All, hit.point, hit.normal);
        }
    }

}

[PunRPC]
0 references
void RPC_Shoot(Vector3 hitPosition, Vector3 hitNormal)
{
    Collider[] colliders = Physics.OverlapSphere(hitPosition, 0.3f);
    if(colliders.Length != 0)
    {
        GameObject bulletImpactObj = Instantiate(bulletImpactPrefab, hitPosition + hitNormal * 0.001f, Quaternion.LookRotation(hitNormal, Vector3.up) * bulletImpactPrefab.transform.rotation); ;
        Destroy(bulletImpactObj, 10f);
        bulletImpactObj.transform.SetParent(colliders[0].transform);

    }
}
```

Hình 3.7 Thuật toán bắn súng

- Hàm này thực chất là Raycast

- Hàm RPC_Shoot() dùng để truyền tia Raycast sang có người chơi khác khi bắn trúng và gây sát thương IEDamagetable().

+ Thuật toán bị hạ gục.

```
public void Die()
{
    PhotonNetwork.Destroy(controller);
    deaths++;
    Invoke("GetDie", 3f);

}
0 references
public void GetDie()
{
    CreateController();
    Hastable hash = new Hastable();
    hash.Add("deaths", deaths);
    PhotonNetwork.LocalPlayer.SetCustomProperties(hash);
    singleGun = new SingleShotGun();
    singleGun.mag = 5;
}
```

Hình 3.8 Thuật toán hạ gục

- Hàm Die() thực chất là Destroy() nhân vật khi bị bắn.

- Sau khi Destroy sẽ sử dụng Invoke để delay hàm GetDie hồi sinh lại nhân vật sau năm giây

- Lúc chết biến Death++ sẽ được cộng dồn để thống kế số lượt bị hạ gục của người chơi.

+ Thuật toán hạ gục đối phương.

```
1 reference
public void GetKill()
{
    PV.RPC(nameof(RPC_GetKill), PV.Owner);
}

[PunRPC]
1 reference
void RPC_GetKill()
{
    kills++;
    Hastable hash = new Hastable();
    hash.Add("kills", kills);
    PhotonNetwork.LocalPlayer.SetCustomProperties(hash);
}
```

Hình 3.9 Thuật toán hạ gục đối phương

Hàm GetKill() sẽ thực thi hàm RPC_GetKill, nhận giá trị xem người bị bắn đã bị hạ gục chưa, biến kills++ sẽ được đếm để thống kê số kill

CHƯƠNG 4: THỰC NGHIỆM VÀ TRIỂN KHAI

1. Giới thiệu hệ thống game

Trò chơi mang lên là Color Gun, thuộc dòng game bắn súng góc nhìn thứ nhất. Trò chơi có 3 màn hình chính là màn hình Intro – Để dẫn vào game, màn hình Lobby – Để lựa chọn các chức năng như tạo phòng, chọn phòng, thoát game và màn hình GamePlay là màn hình diễn ra trận đấu. Trò chơi cho phép người chơi bắn với nhau, chiến đấu với nhau và gạ gục nhau.

2. Hệ thống game

+ **Màn hình Intro.**

Hình 4.1 Màn hình Intro

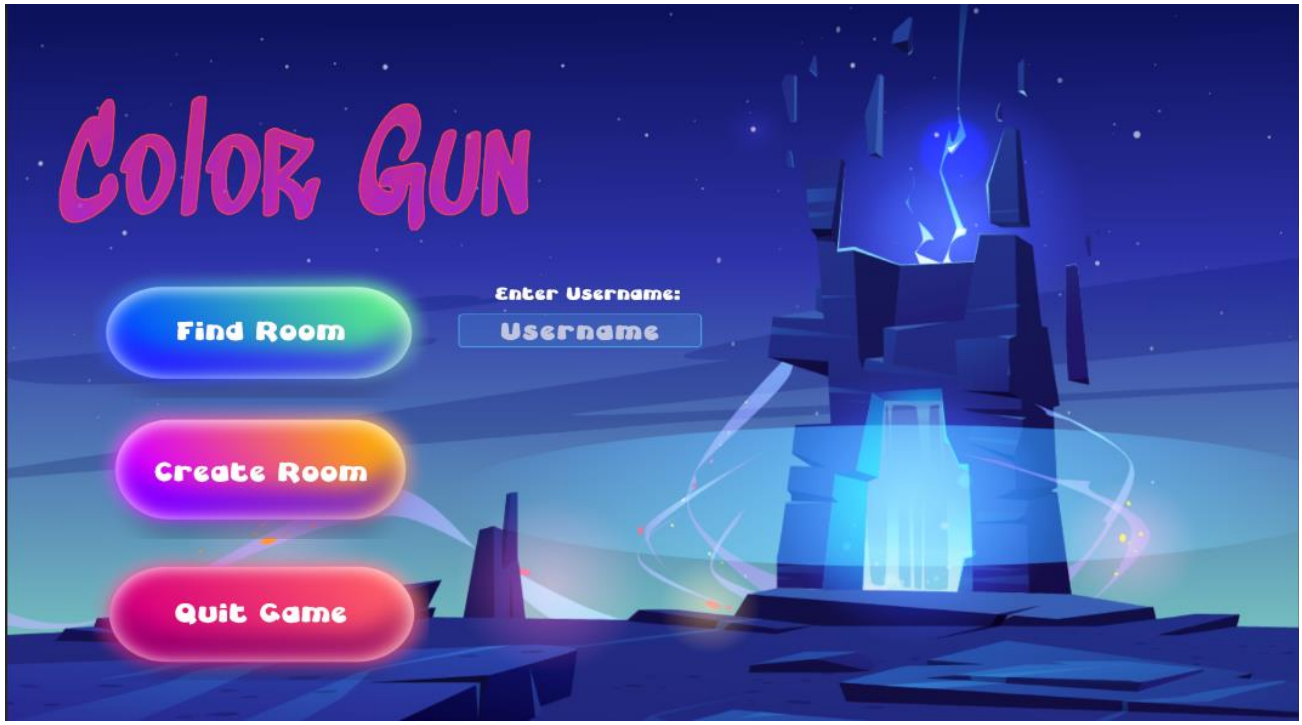
+ **Màn hình Lobby**

Màn hình Lobby gồm năm thành phần

- Loading – Là màn hình hiển thị khi game đang load dữ liệu

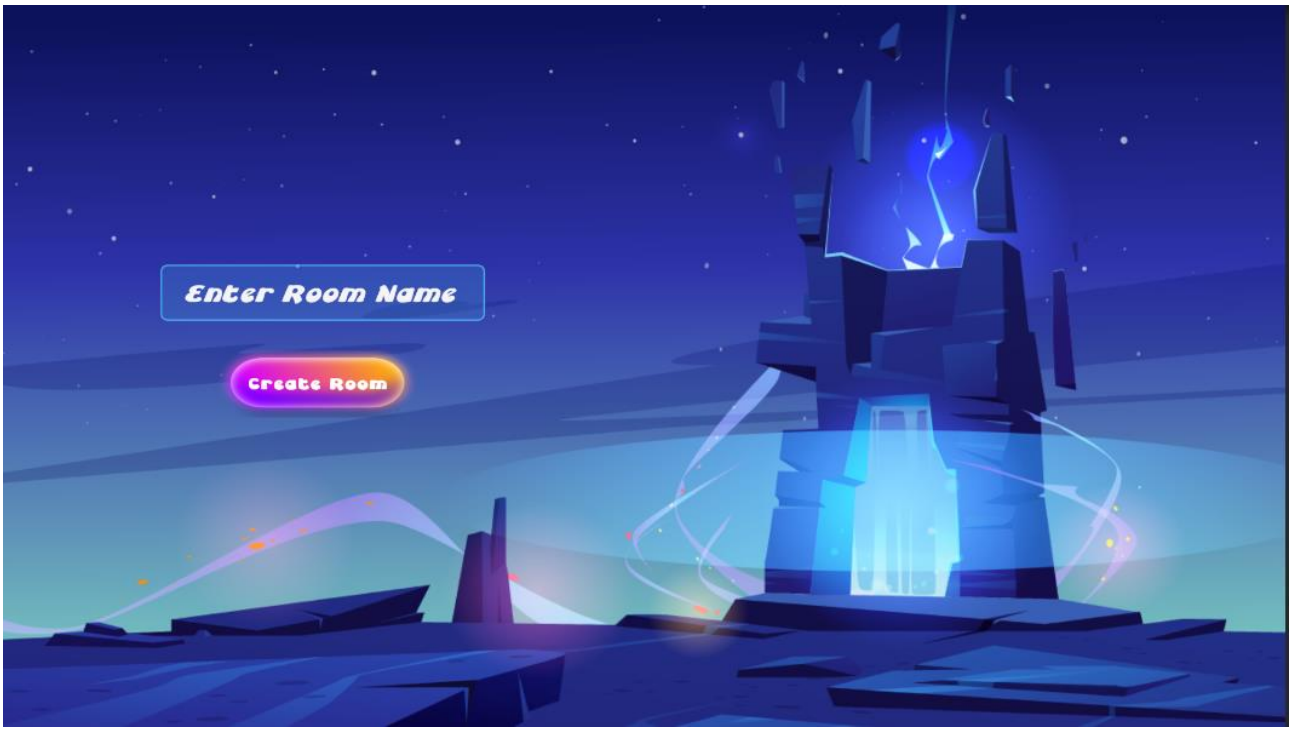
Hình 4.2 Loading game

- Main menu – Là menu chính để người chơi lựa chọn chức năng, ở đây có phần nhập tên người chơi, tên này sẽ hiển thị bên trong game phân biệt với người chơi khác.



Hình 4.3 Main menu

- Create Room Menu – Là nơi để người chơi nhập tên phòng muốn tạo và nhấn Create Room để tạo phòng.



Hình 4.4 Create Room

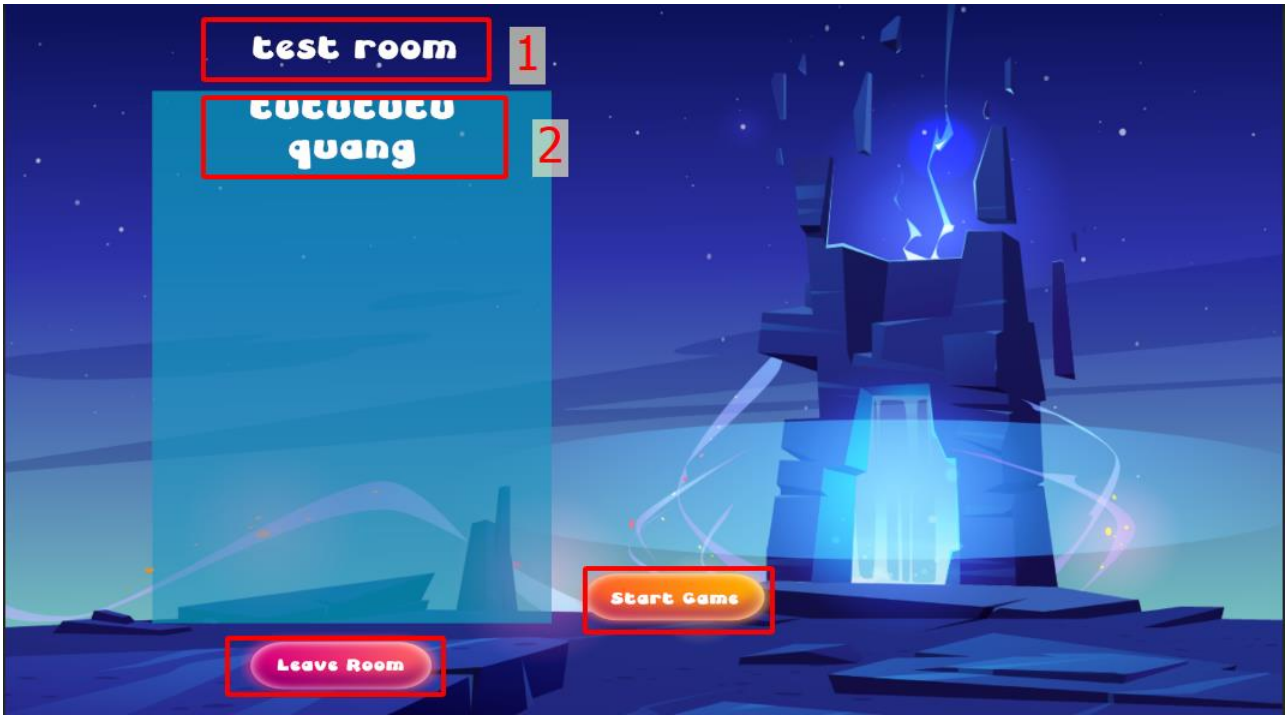
- Room menu – Là nơi sau khi tạo phòng, hiển thị danh sách người chơi có trong phòng và có nút Start để vào game, phòng chơi có các thành phần sau

[1] là tên của phòng chơi, ở đây là **test room**

[2] là nơi hiển thị các người chơi có trong phòng, ở đây hiện tại là 2 người.

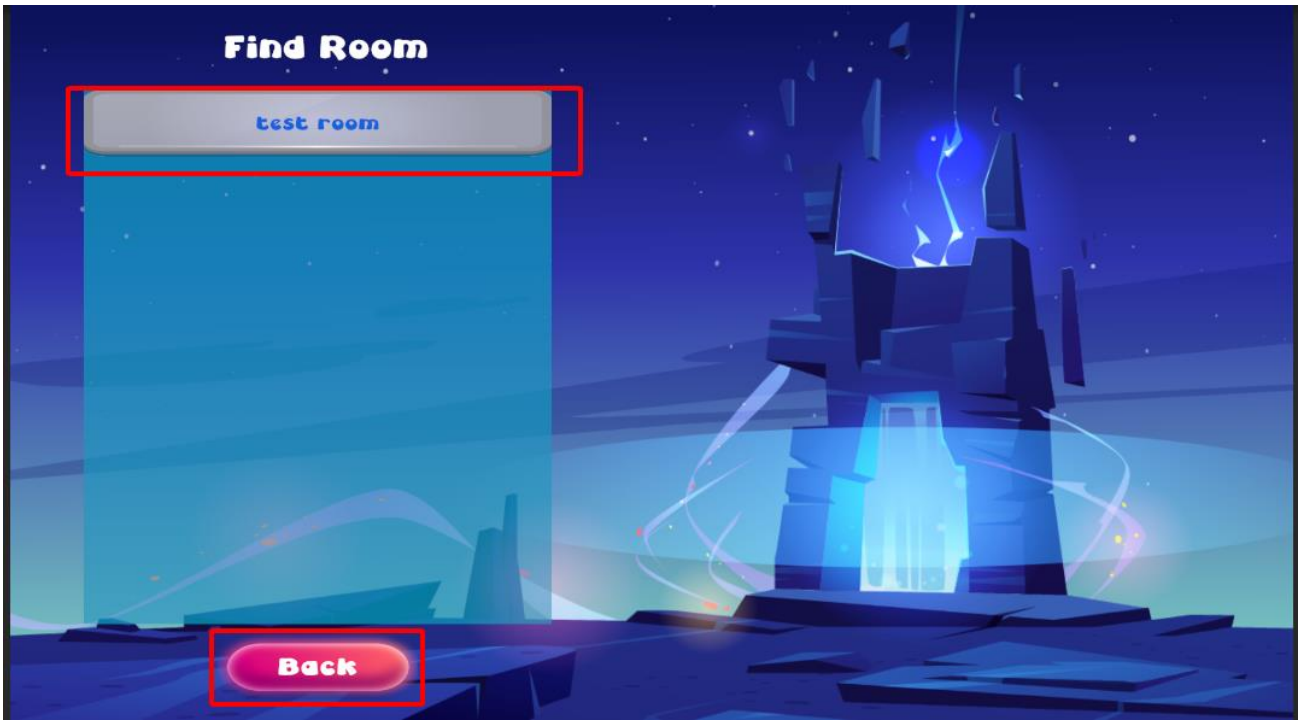
Nút Start để bắt đầu game.

Nút Leave để thoát khỏi phòng.



Hình 4.5 Room menu

- Find Room menu – Là nơi hiển thị các phòng để người chơi có thể chọn và tham gia. Hiện tại ở đây đang có một phòng chơi là **test room** và nút back là để quay trở về.



Hình 4.6 Find room menu

+ **Màn hình GamePlay**

Hình 4.7 Màn hình gamePlay qua góc nhìn của nhân vật

Ảnh trên là màn hình game Play qua góc nhìn của nhân vật sẽ có 3 thứ chính

[1] là thanh sinh lực của nhân vật, nếu thanh sinh lực này cạn thì nhân vật sẽ bị hạ gục, cách làm giảm sinh lực là bị đối phương bắn

[2] Là hiển thị số lượng đạn và số đạn hiện tại, nếu số đạn hiện tại là 0 thì súng sẽ không bắn được nữa và phải ấn R để thay đạn

[3] Là vũ khí nhân vật đang trang bị, sẽ có 2 vũ khí là súng lớn và súng lục. Vũ khí sẽ bắn và làm giảm sinh lực của đối phương

Hình 4.8 Màn hình GamePlay từ trên nhìn xuống

Ảnh trên là màn hình game Play qua góc nhìn từ trên xuống gồm 2 thứ.

[1] là nhân vật đang đứng ở vị trí trên bản đồ

[2] là hòm máu, khi nhân vật chạm vào thì thanh sinh lực của nhân vật sẽ đầy, trên map chỉ có một hòm máu nên người chơi phải cân nhắc tránh bị đối phương bắn, tạo thách thức cho trò chơi

Hình 4.9 Màn hình kết quả

Ảnh trên là màn hình kết quả, hiện ra bằng cách giữa nút Tab, nó sẽ hiển thị trong phòng hiện đang có bao nhiêu người và số lần hạ gục hay bị hạ gục của mỗi người chơi. Người chơi có thể nhìn vào đó để phân thắng bại.

3. Triển khai

3.1 Cấu hình máy

Game được xây dựng trên chiếc máy tính có cấu hình như sau

- Bộ xử lý: AMD Ryzen 5 4000 series 3.00GHZ
- Ram: 16 GB
- Đồ họa: Nvidia GTX 1650 Ti
- Ổ cứng: 1TB.

3.2 Hệ điều hành

- Game được xây dựng cho hệ điều hành : Windows 7, Windows 10 và Windows
- Game được xây dựng bằng Unity chạy trên Windows 11.

3.3 Các phần mềm và framework liên quan.

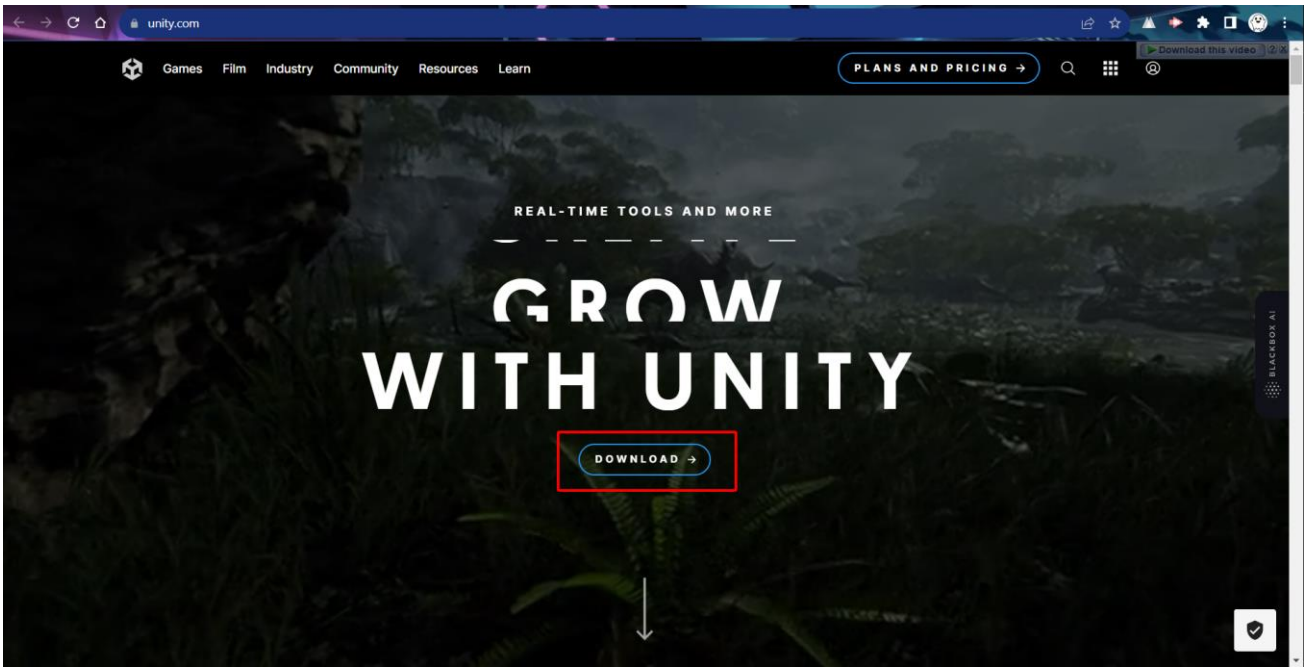
- Engine được sử dụng là Unity với phiên bản Unity Editor 2021.3.12f1

- Các package được sử dụng trong Unity
    + PhotonUnityNetwork
    + FPS animation framework
    + Scifi – gun – pack
    + JMO Assets
    + LowPolyFPSLite

4. Cách thức cài đặt

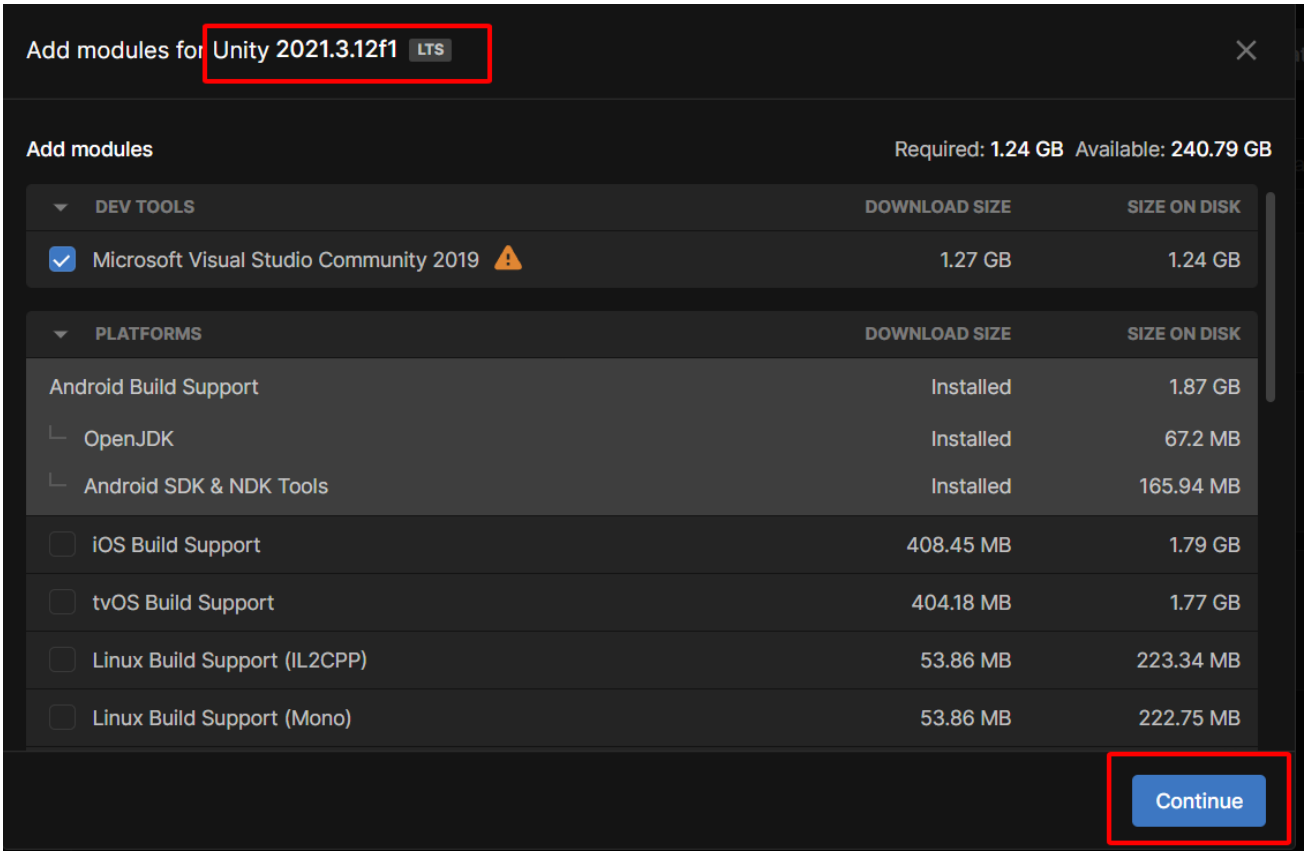
4.1 Chạy game bằng Unity

B1: Ta cài đặt Unity Hub tại trang Web https://unity.com/



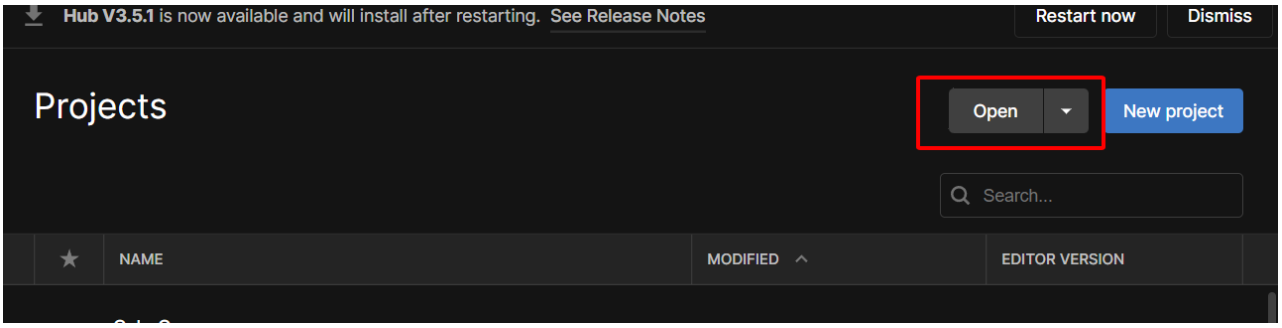
Hình 4.10 Tải Unity Engine

B2: Sau khi cài đặt xong Unity Hub, ta đăng nhập và chọn tải Unity Editor phiên bản 2021.3.12f1

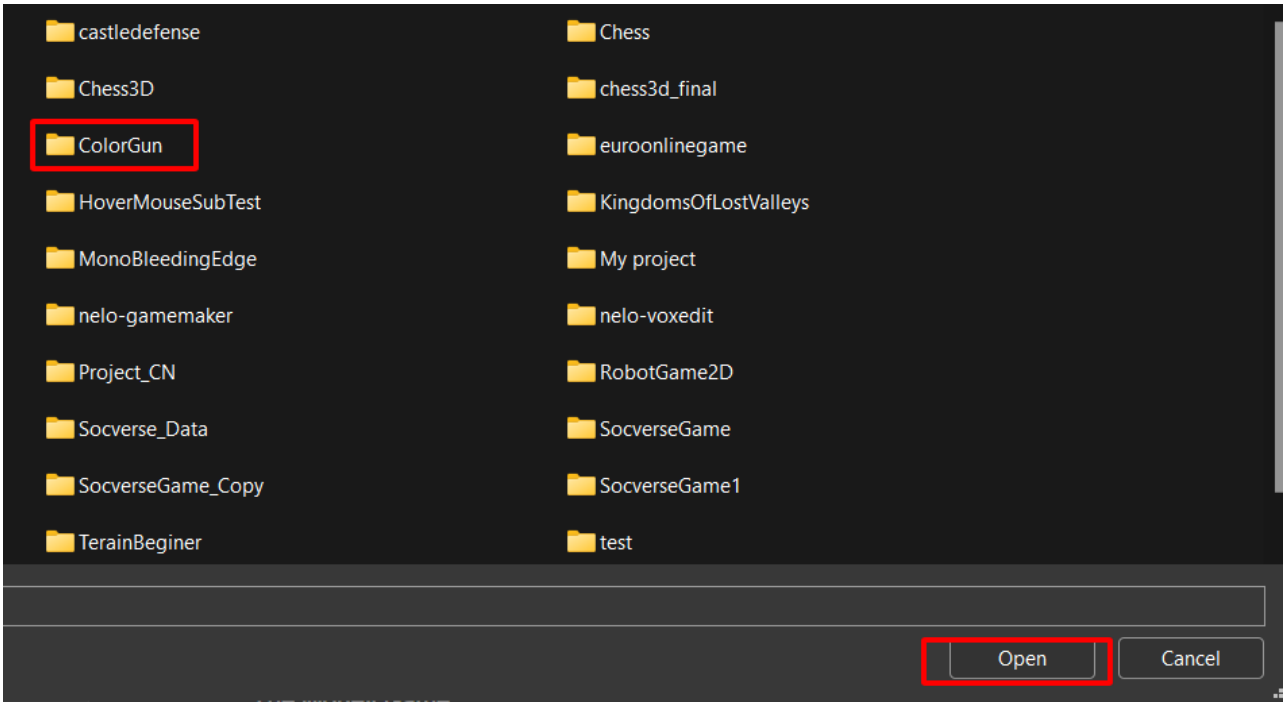


Hình 4.11 Tải Unity Editor phiên bản 2021.3.12f1

B4: Sau khi cài xong, ta mở Unity Hub, chọn Open và chọn Project muốn mở, ở đây là thư mục ColorGun và chờ đợi Unity mở.

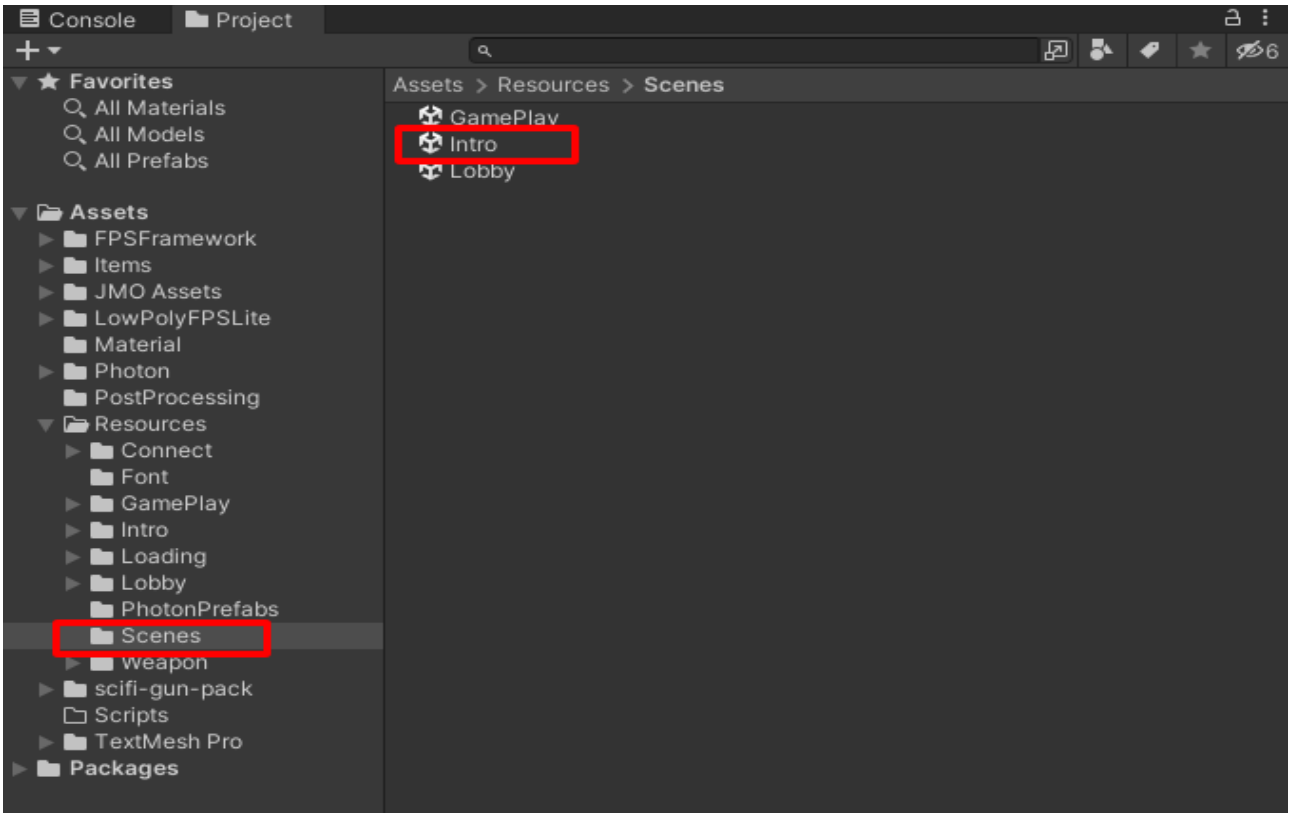


Hình 4.12 Mở project



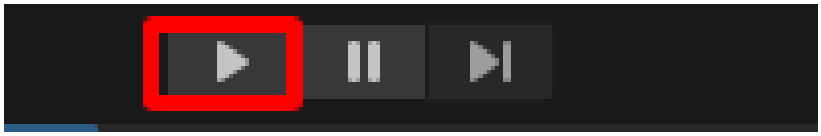
Hình 4.13 Chọn thư mục ColorGun

B5: Sau khi đã mở xong game, ở mục Project ta chọn thư mục tên Scene và chọn Intro.



Hình 4.14 Chọn Scene Intro

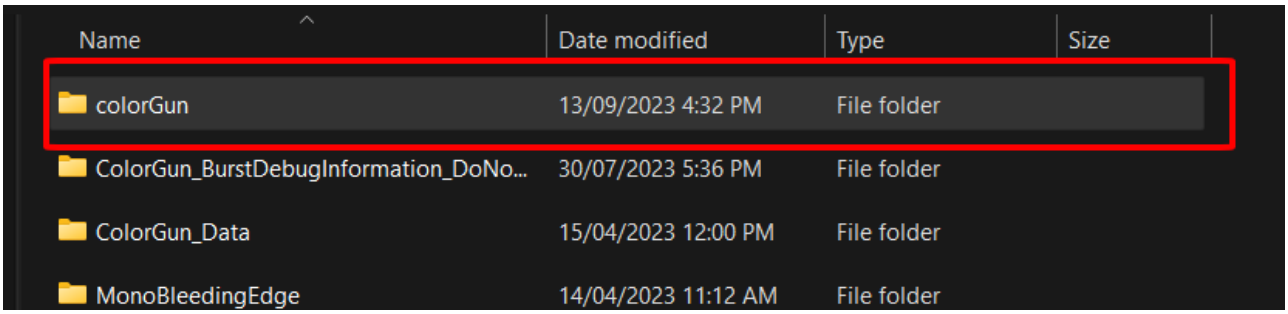
B6: Ta nhấn nút Play bên trên cùng và vào game.



Hình 4.15 Ấn nút play

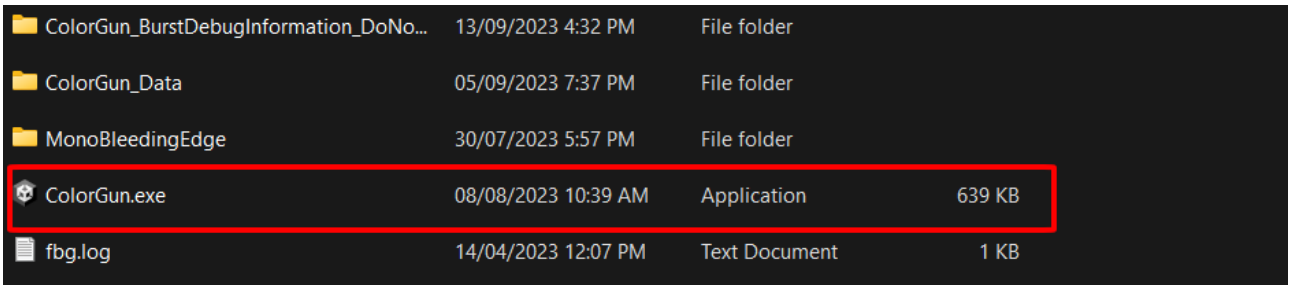
4.2 Chạy game bằng file .exe

B1 Vào thư mục ColorGun



Hình 4.16 chọn thư mục Color Gun

B2 Nháy đúp vào ColorGun.exe và chơi



Hình 4.17 Chọn ColorGun.exe

Sau khi cài đặt game và chạy game thì ta sẽ thưởng thức tựa game.

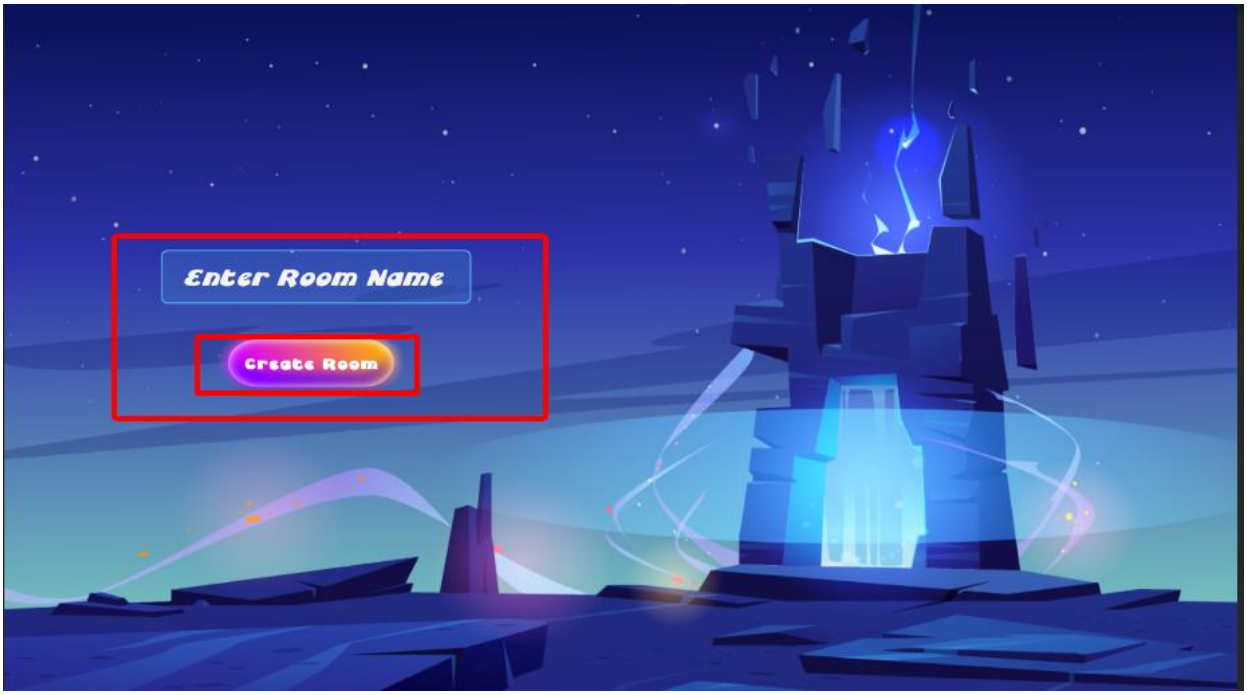
5. Cách thức chơi game

Ở màn hình Main menu ta sẽ nhập tên ở mục User name

Hình 4.18 Hướng dẫn chơi game Main menu

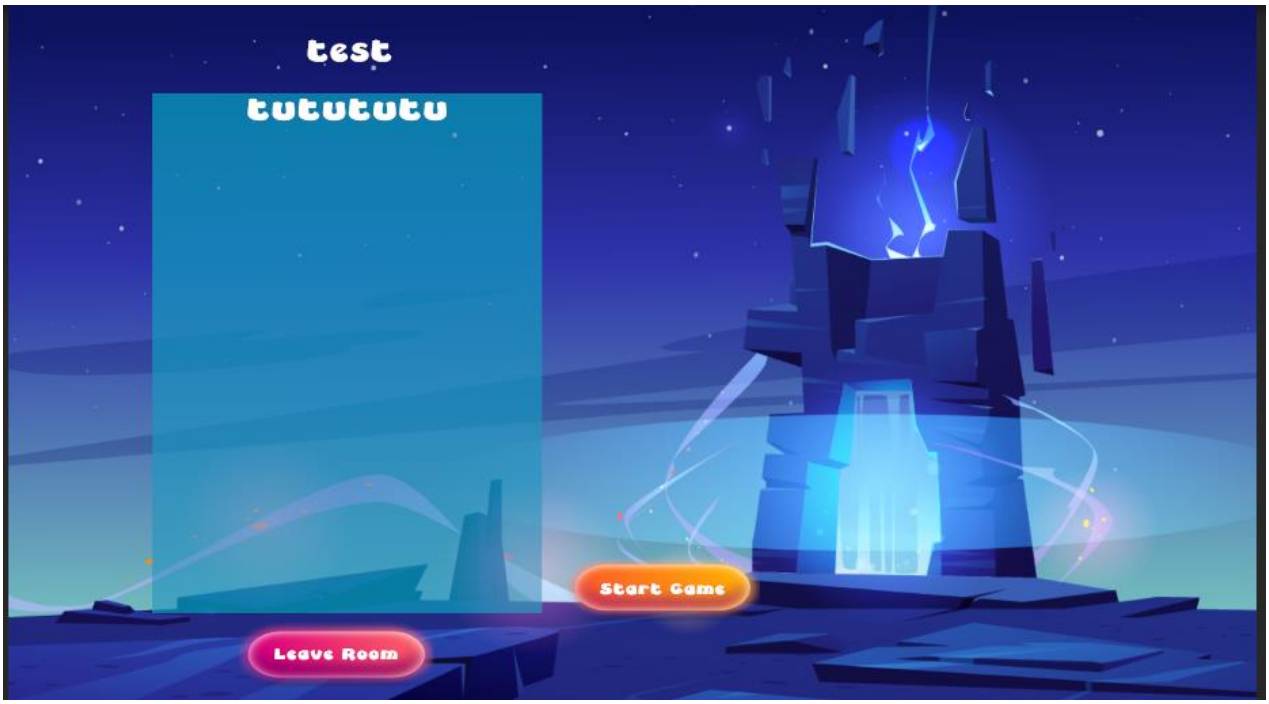
- Tạo phòng Create Room

Tiếp theo nếu người chơi muốn tạo một phòng mới ta sẽ ấn Create Room. Màn hình Create Room hiện ra để người chơi nhập tên mà mình muốn sau đó nhấn Create.



Hình 4.19 Hướng dẫn chơi game Create Room

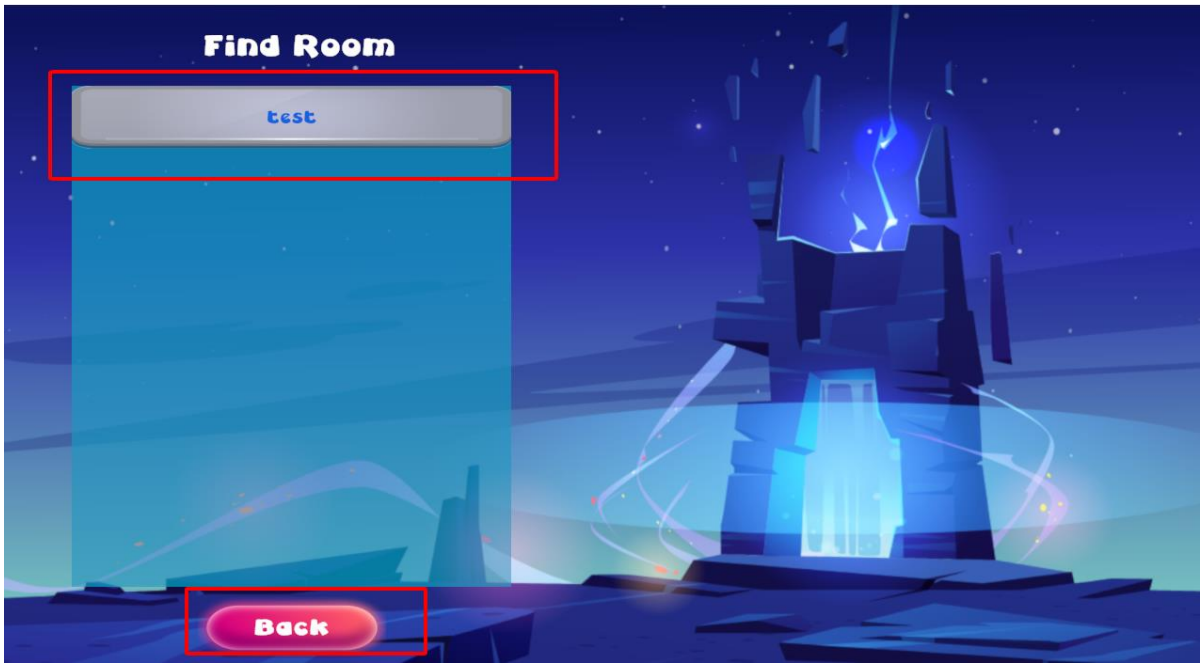
Tiếp theo nếu người chơi sau khi ấn Create thì giao diện phòng hiện ra, việc còn lại đó chính là đợi người chơi khác tham gia phòng, sau khi đã có đủ người mình mong muốn, ta nhấn Start Game để bắt đầu chơi



Hình 4.20 Hướng dẫn chơi game Create Room Menu

- Tham gia phòng Find Room

Tại Main menu ta chọn nút Find Room, sau đó màn hình danh sách phòng đã tạo sẽ hiện ra, nếu người chơi muốn tham gia phòng nào thì sẽ ấn vào phòng đó, sẽ chuyển tới phòng nhưng khác ở việc tạo phòng là không có nút Start Game vì chủ phòng mới được quyền Start Game.



Hình 4.21 Hướng dẫn chơi game Find Room

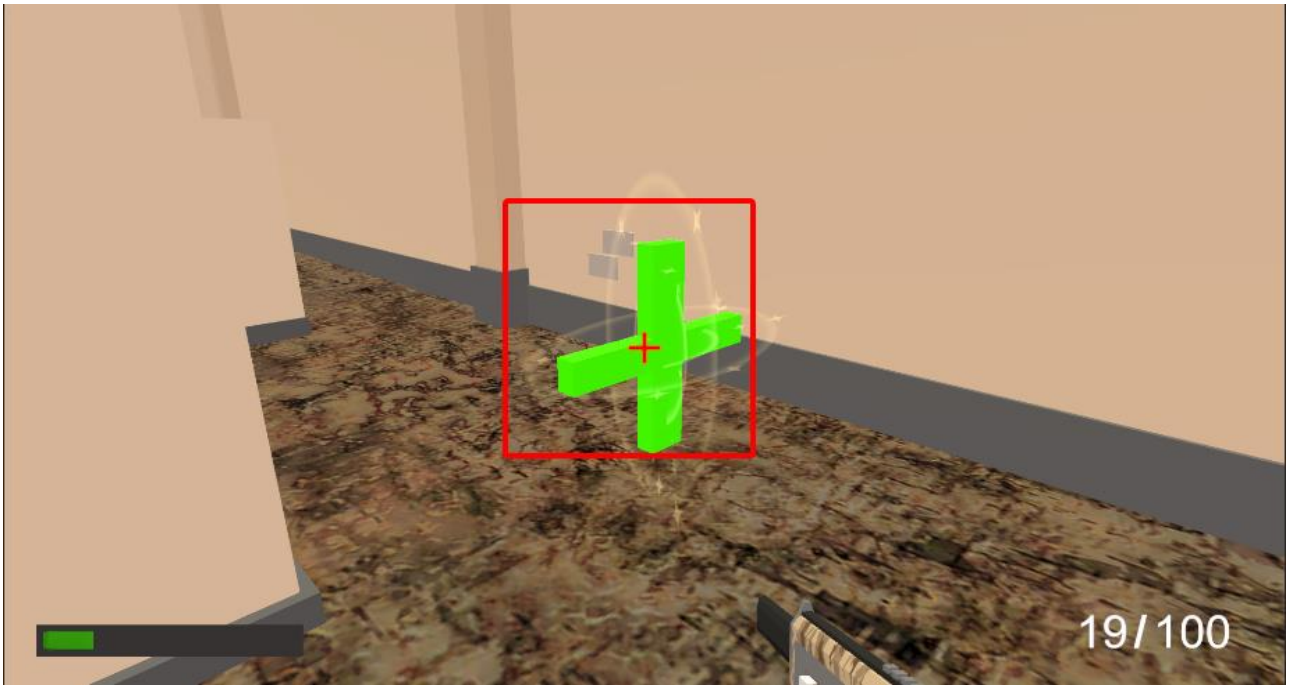
- Các nút điều khiển chơi game và cách thức chơi
    + Sau khi Start Game, người chơi sẽ được vào màn GamePlay. Ở đây người chơi sẽ di chuyển lên, xuống, trái, phải bằng nút W, S, A, D.
    + Người chơi nhảy bằng nút SPACE và quay 360 độ bằng cách quay chuột
    + Người chơi có thể chạy bằng cách giữ nút SHIFT và đè W
    + Mục tiêu của người chơi là người chơi khác, nhắm kỹ tầm ngắm vào đối phương và ấn chuột trái để bắn.

Hình 4.22 Nhắm vào đối thủ

+ Khi bị đối thủ bắn thì thanh sinh lực sẽ rút ngắn dần, rút ngắn hết sẽ bị hạ gục

Hình 4.23 Thanh sinh lực bị ngắn dần

+ Lúc đó, người chơi có thể đứng cạnh biểu tượng chữ thập để có thể hồi đầy thanh lại



Hình 4.24 Biểu tượng hồi máu

+ Người chơi có thể cài đặt độ nhạy của chuột, độ lớn của âm thanh bằng cách ấn ESC, điều chỉnh cách thanh trượt, thoát game bằn cách ấn Leave và trở lại chơi game bằng nút Back

Hình 4.25 Cài đặt

+ Người chơi có thể xem thống kê kết quả bằng cách giữ Tab, màn hình kết quả sẽ hiện ra

Hình 4.26 Thống kê kết quả

KẾT LUẬN

Thuận lợi

+ Được sự hướng dẫn nhiệt tình, tận tâm từ người hướng dẫn chuyên đề.
+ Được tiếp thu, tham khảo kiến thức từ các nguồn trên internet.

Khó khăn

+ Kinh nghiệm và kiến thức còn hạn chế
+ Còn mới mẻ trong việc tiếp thu kiến thức mới

Kết quả đạt được và hướng phát triển

Kết quả đạt được

+ Vận dụng được các thư viện từ Unity
+ Học được cách xoay chuyển thời gian để phù hợp hơn với công việc
+ Học được cách sử dụng công nghệ mới là Photon Unity Network (PUN)
+ Xây dựng được game bắn súng góc nhìn thứ nhất (ColorGun) từ bước thiết kế tới khi hoàn chỉnh.

Hướng phát triển

+ Phát triển thêm các chức năng mới để cho người chơi cảm thấy thú vị hơn như súng bắn liên thanh, ăn hòm đạn, ngồi bắn, …
+ Cải thiện về đồ họa game từ **Non – poly** thành **real graphic**.

DANH MỤC TÀI LIỆU THAM KHẢO

[1] Lập trình Unity là gì và những kiến thức tổng quan cần nắm vững, link: https://itnavi.com.vn/blog/lap-trinh-unity

Ngày tham khảo: 08/2023.

[2] Jeff W. Murray (2021), C# Game Programming Cookbook for Unity 3D, nhà xuất bản CRC Press, Mỹ.

Ngày tham khảo 06/2023.

[3] Photon Unity Networking: General Documentation, link: https://documentation.help/Photon-Unity-Networking-1.91/general.html

Ngày tham khảo: 06/2023.

[4] HOW TO CREATE AN ONLINE MULTIPLAYER GAME WITH UNITY, link: https://paladinstudios.com/2013/07/10/how-to-create-an-online-multiplayer-game-with-unity/

Ngày tham khảo: 07/2023