MỤC LỤC

DANH MỤC HÌNH VẼ

Chương 1 . TỔNG QUAN

1.1 Giới thiệu Unity

Một hệ sinh thái game gồm có các chức năng cơ bản như: cung cấp công cụ dựng hình (kết xuất đồ họa) cho các hình ảnh 2D hoặc 3D, công cụ vật lý (tính toán và phát hiện va chạm), âm thanh, mã nguồn, hình ảnh động, trí tuệ nhân tạo, phân luồng, tạo dò nguồn dữ liệu xử lý, quản lý bộ nhớ, dựng ảnh đồ thị và kết nối mạng.Ngoài những chức năng cơ bản của một hệ sinh thái đó, Unity còn có những ưu việt vượt trội so với các engine khác:

- **Ngôn ngữ lập trình phổ biến Việt Nam**: C#.
- **Hỗ trợ đa nền tảng**: Lập trình viên dùng Unity3D engine và ngôn ngữ C# hoặc script để phát triển game hoàn thiện, sau đó Unity cho phép bạn "build" ra các phiên bản cho các nền tảng khác mà không cần viết thêm dòng code nào, giúp bạn rút ngắn rất nhiều thời gian xây dựng game cũng như nâng cao mức độ an toàn khi lập trình game. Những nền tảng mà Unity đang hỗ trợ gồm PlayStation 3, Xbox 360, Wii U, iOS, Android, Windows, Blackberry 10, OS X, Linux, trình duyệt web.
- **Dễ sử dụng, ngay cả với Lập trình viên nghiệp dư**, do Unity3D được xây dựng trong một môi trường phát triển tích hợp, cung cấp một hệ thống toàn diện cho các lập trình viên, từ soạn thảo mã nguồn, xây dựng công cụ tự động hóa đến trình sửa lỗi.
- **Tính kinh tế cao**: Những cá nhân và doanh nghiệp có doanh thu dưới 100.000 USD/năm được dùng miễn phí engine Unity3D, và Unity Technology chỉ thu phí 1.500 USD/năm cho bản Pro- một con số rất khiêm tốn so với những gì engine này mang lại.
- **Rất được ưa chuộng tại Việt Nam**, ngay cả trong các game studio lớn như VTC, VNG, Glass-Egg.

- **Thư viện phong phú, đa dạng**: Unity có nhiều thư viện, các công cụ hỗ trợ làm game nhanh hơn, thông minh hơn, các đối tượng được tạo sẵn, và tất cả các thư viện này đều "mở", cho phép cộng đồng tự do sử dụng và sáng tạo nên các sản phẩm của chính mình, thậm chí có thể bán trên Asset Store của Unity.
- **Cộng đồng rất lớn mạnh**: là engine phổ biến nhất trên thế giới, Unity có cộng đồng sử dụng rất lớn mạnh. Mọi thắc mắc của bạn về Unity đều sẽ được trả lời trên website cộng đồng http://answers.unity3d.com.
- **Hỗ trợ Networking để phát triển MMO game**.

1.2 Mục tiêu đề tài

Đề tài này thuộc hướng tìm hiểu công nghệ từ đó xây dựng ứng dụng. Mục tiêu của đề tài là tìm hiểu engine Unity và sử dụng Unity xây dựng thử nghiệm game thể loại phiêu lưu kinh dị (game adventure) Để thực hiện được được điều này nội dung của đồ án bao gồm: Tìm hiểu tổng quan về kiến trúc của Unity và cách tạo lập các ứng dụng trong Unity.Tìm hiểu các vấn đề như load mô hình (scenes) vào game, làm nhân vật chuyển động,cách tạo địa hình, giao diện, âm thanh và các hiệu ứng particle…để rồi từ đó đưa ra giải pháp.Xây dựng và phát triển ứng dụng game thể phiêu lưu kinh dị bằng Unity.

1.3 Kế hoạch thực hiện

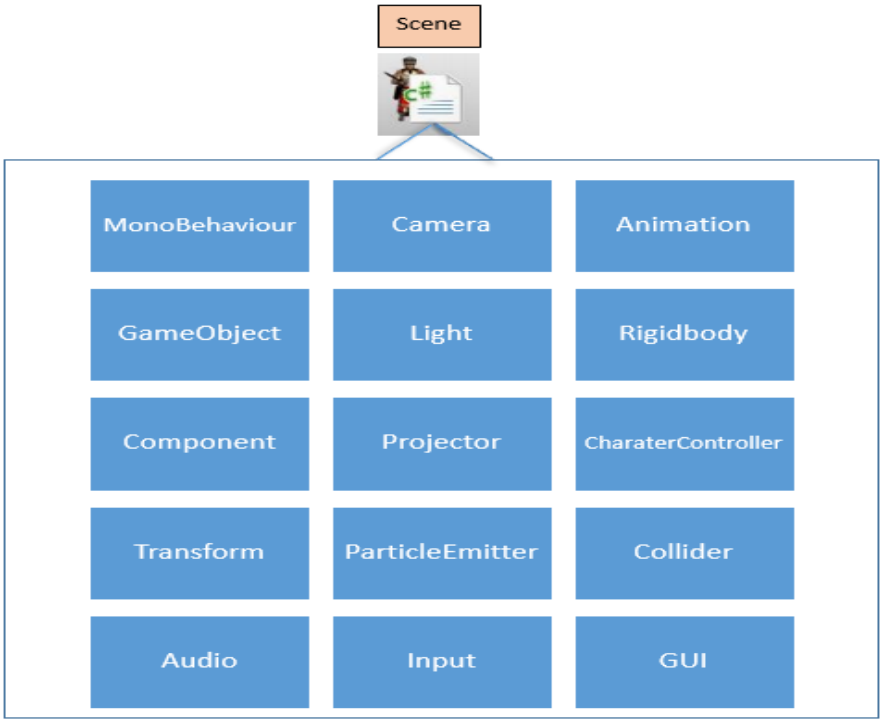
- Tuần 1,2: Chọn đề tài và tìm hiểu nền tảng lập trình trên Unity
- Tuần 3,4: Tìm hiểu cách lập trình game với Unity và thư viện.
- Tuần 5: Tìm hiểu các Game Engine hỗ trợ làm game với Unity
- Tuần 6 : Tìm hiểu các phương pháp xây dựng game với Unity.
- Tuần 7,8: Mô tả đặc trưng kiến trúc và nội dung game.
- Tuần 9,10 : Xây dựng game.

Chương 2 . CƠ SỞ GAME ENGINE

2.1 Kiến trúc tổng quan

Engine Unity hỗ trợ cho chúng ta UnityAPI để viết script game. UnityAPI là API lập trình game trong Unity rất mạnh.UnityAPI chứa các đối tượng và phương thức trỗ trợ hầu hết các đối tượng và các loại thành phần trong Unity.

Trong một scene thường có nhiều đối tượng game.Mỗi đối tượng này có thể có hoặc không có đoạn script nào gắn lên đó,Nếu muốn gắn script vào đối tượng, ta bắt buộc phải kế thừa class đó từ lớp MonoBehaviour của UnityAPI và tên class phải trùng vơi tên file script. Mỗi script khi gắn lên đối tượng game đều được đối tượng game xem như một như một thành phần bên trong và được cấp phát vùng nhớ khi chạy game.



Hình 1. Kiến trúc tổng quan Unity

Bên trong UnityAPI chứa rất nhiều lớp hỗ trợ lập trình game, trong đó có một số lớp quan trọng như :

MonoBehaviour: tất cả các script muốn gắn vào một đối tượng game bắt buộc phải kế thừa từ lớp này.

GameObject: lớp cha của tất cả các thực thể trong scene.

Component: lớp cha của tất cả các thành phần có thể gắn vào đối tượng.

Transform: giúp thay đổi vị trí, xoay, biến đổi tỉ lệ mô hình.

Input: hỗ trợ lập trình với chuột, cảm ứng đa điểm, cảm biến gia tốc.

Light: giúp tạo ánh sáng trong game.

Projector: giúp chiếu texture lên bề mặt vật thể.

ParticleEmitter: hỗ trợ tạo các hiệu ứng particle đẹp mắt.

Audio: hỗ trợ lập trình với âm thanh.

Animation: chạy chuyển động của mô hình nhân vật.

Rigidbody: giúp tạo hiệu ứng vật lý liên quan đến trọng lực như bóng nẩy, lăn,…

CharacterController: giúp điều khiển nhân vật di chuyển theo độ cao địa hình.

Collider: hỗ trợ lập trình va chạm giữa các vật thể.

GUI: giúp lập trình giao diện người dùng trên Unity.

Camera: giúp lập trình camera.

2.2 Các đặc điểm và tính năng của Unity

2.2.1 *Rendering (kết xuất hình ảnh)*

Giống như tất cả các Engine hoàn chỉnh khác, Unity hỗ trợ đầy đủ khả năng kết xuất hình ảnh (Redering) cùng nhiều hỗ trợ cho phép áp dụng các công nghệ phổ biến trong lĩnh vực đồ họa 3D nhằm cải thiện chất lượng hình

ảnh. Các phiên bản gần đây nhất của Unity được xây dựng lại thuật toán nhằm cải thiện hiệu suất kết xuất hình ảnh đồng thời tăng cường chất lượng hình ảnh sau khi kết xuất.

Một số hỗ trợ:

Unity cung cấp sẵn 100 Shaders với đầy đủ các loại phổ biến nhất.

Hỗ trợ Surface Shaders, Occlusion Culling, GLSL Optimizer.

Hỗ trợ LOD.

2.2.2 *Lighting (ánh sáng)*

Ánh sáng là một điều thiết yếu giúp môi trường trở nên đẹp và thực tế hơn. Unity cũng cung cấp nhiều giải pháp đa dạng cho phép chúng ta áp dụng ánh sáng một cách tốt nhất vào môi trường trong trò chơi với nhiều loại nguồn sáng như ánh sáng có hướng (Directional Light), ánh sáng điểm (Point Light), ... Một số công nghệ và kỹ thuật về ánh sáng được Unity hỗ trợ: Lingtmapping, Realtime Shadows, hiệu ứng Sunshafts và Lens Flares.

2.2.3 *Terrains (địa hình)*

Terrains còn gọi chung là địa hình bao gồm phần đất nền của môi trường trong trò chơi cùng các đối tượng gắn liền như cây, cỏ, …

Unity cung cấp một công cụ hỗ trợ rất tốt khả năng này với tên gọi là Terrains Tools cho phép chúng ta thiết kế địa hình với các công cụ vẽ dưới dạng Brush có nhiều thông số tùy chỉnh để tạo hình và lát Texture cho địa hình. Cùng với Terrain Tools là Tree Creator, một công cụ mạnh mẽ cho phép chúng ta tạo ra cây cối với hình dạng, kích thước và kiểu cách đa dạng.

2.2.4 *Substances (Texture thông minh)*

Substances có thể hiểu đơn giản là một dạng tùy biến Textures nhằm làm đa dạng chúng trong nhiều điều kiện môi trường khác nhau. Unity cung

cấp khả năng này thông qua các API dựng sẵn trong thư viện, hỗ trợ lập trình viên lập trình để tùy biến hình ảnh được kết xuất của Texture

2.2.5 *Physics (vật lí)*

PhysX là một Engine mô phỏng và xử lí vật lý cực kỳ mạnh mẽ được phát triển bởi nhà sản xuất card đồ họa hàng đầu thế giới NVIDIA. Unity đã tích hợp Engine này vào để đảm nhận mọi vấn đề vật lý. Một số vấn đề vật lý được hỗ trợ bởi Unity như: Soft Bodies, Rigitbodies, Ragdolls, Joints, Cars, …

2.2.6 *Pathfinding (tìm đường)*

Đây là một tính năng rất mới mẻ đến từ phiên bản Unity 3.5. Với các phiên bản trước, để phát triển khả năng tìm đường cho trí thông minh nhân tạo (AI), nhà phát triển phải hoàn toàn tự xây dựng cho mình một hệ thống tìm đường riêng biệt. Tuy nhiên ở phiên bản 3.5 đến nay, Unity hỗ trợ cho chúng ta tính năng Pathfinding cho phép tạo ra khả năng tìm đường cho AI nhờ vào khái niệm lưới định hướng (NavMesh).

2.2.7 *Audio (âm thanh)*

Về âm thanh, Unity tích hợp FMOD – công cụ âm thanh thuộc hàng mạnh nhất hiện nay. Qua đó Unity hỗ trợ chúng ta nhập và sử dụng nhiều định dạng tập tin âm thanh khác nhau.

2.2.8 *Programming (lập trình)*

Lập trình là một trong những yếu tố quan trọng nhất trong phát triển Game. Lập trình cho phép nhà phát triển tạo nên khả năng tương tác, trí thông minh và yếu tố Gameplay cho trò chơi.

Unity cho phép chúng ta lập trình bằng nhiều ngôn ngữ mạnh mẽ và phổ biến với các lập trình viên như: C#, Java Scrip và Boo.

2.2.9 *Networking*

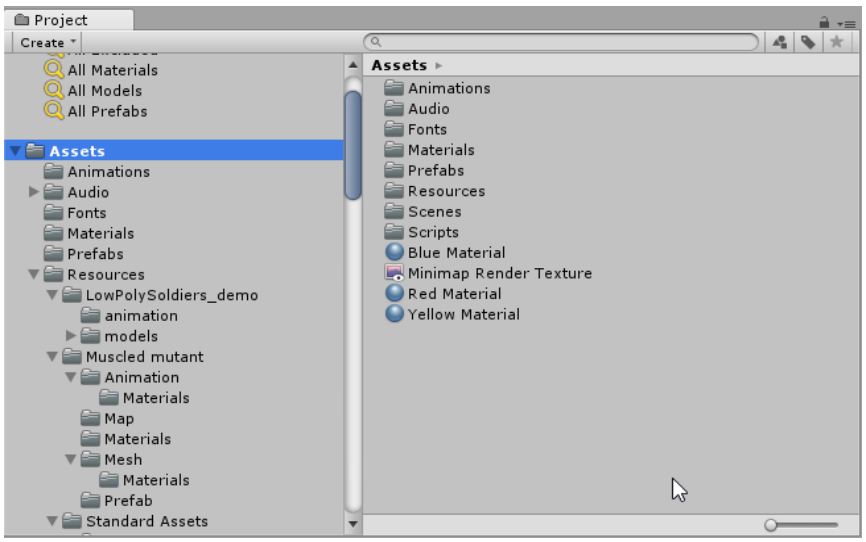
Networking cho phép chúng ta tạo ra các trò chơi trực tuyến (online) – một trong những thể loại trò chơi thu hút được nhiều người chơi nhất. Tính năng này sẽ hỗ trợ đầy đủ để chúng ta tạo nên các khía cạnh phổ biến trong Game online như hệ thống điểm kinh nghiệm, chat và tương tác thời gian thực, …

Một số tính tăng cung cấp bởi Networking như: State Synchronization, Realtime Networking, Remote Procedure Calls, Backend Connectivity, Web Browser Integration, Web Connectivity

2.3 Các thành phần trong Unity

2.3.1 *Assets*

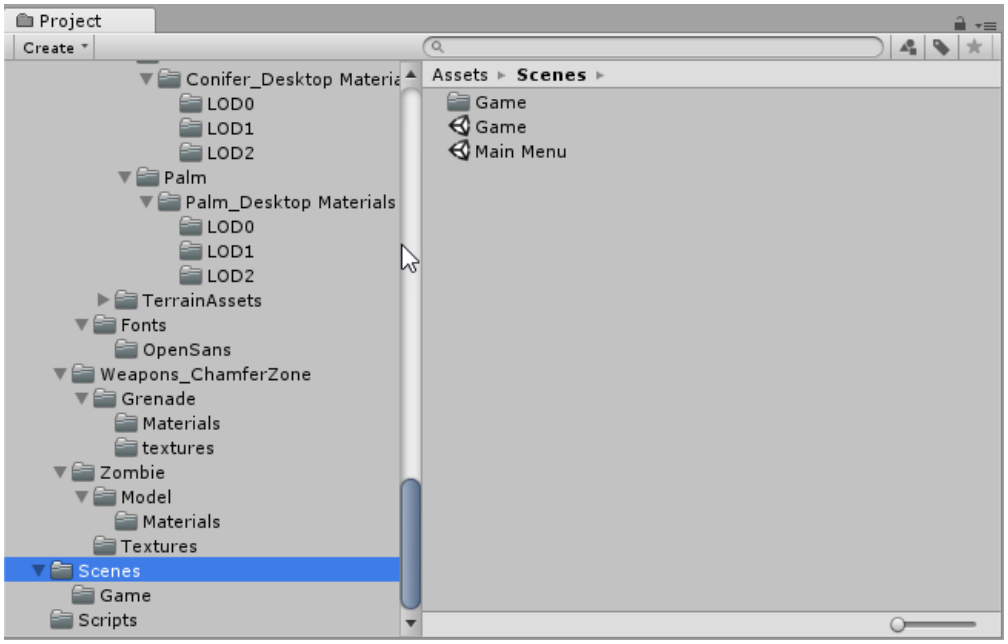
Assets là những tài nguyên xây dựng nên một dự án Unity. Từ những tập tin hình ảnh, mô hình 3D đến các tập tin âm thanh. Unity gọi các tập tin mà chúng ta dùng để tạo nên trò chơi là tài sản (Assets). Điều này lí giải tại sao tất cả các tập tin, thư mục của các dự án Unity đều được lưu trữ trong một thư mục có tên là "Assets".



Hình 2. Thư mục Assets.

2.3.2 *Scenes*

Trong Unity, chúng ta cần hiểu một cảnh (hay một phân đoạn) nghĩa là một màn chơi riêng biệt hoặc một khu vực hay thành phần có trong nội dung của trò chơi (ví dụ như Game menu). Bằng cách tạo nên nhiều Scenes cho trò chơi, chúng ta có thể phân phối thời gian tải hoặc kiểm tra các phần khác nhau của trò chơi một cách riêng lẽ.

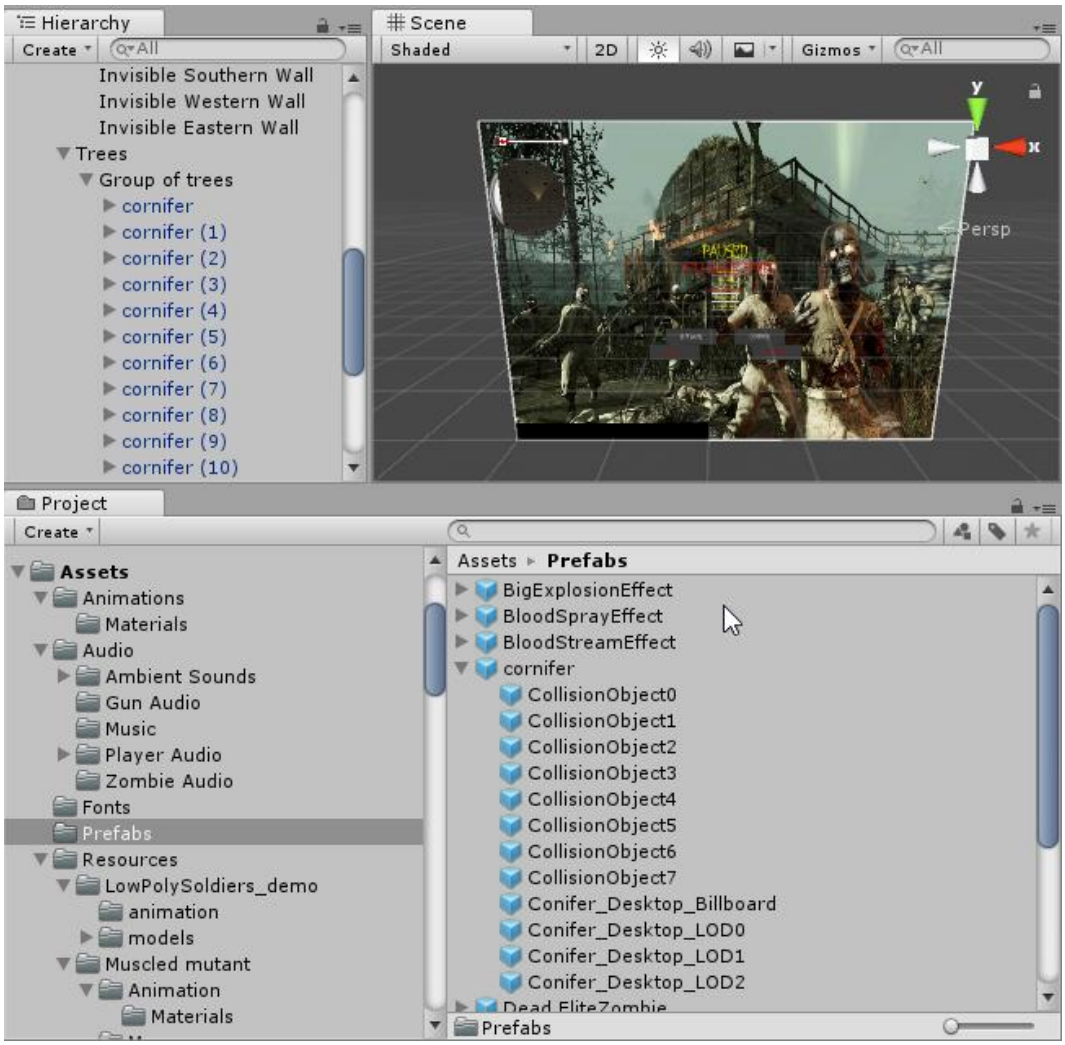


Hình 3. Các scenes của Unity.

2.3.3 *Game Object*

Khi Assets được sử dụng trong Scene, chúng trở thành Game Object một thuật ngữ được sử dụng trong Unity (đặc biệt là trong mảng lập trình). Tất cả các Game Object đều chứa ít nhất một thành phần là Transform. Transform là thông tin về vị trí, góc xoay và tỉ lệ của đối tượng, tất cả được mô tả bởi bộ 3 số X, Y, Z trong hệ trục tọa độ . Thành phần này có thể được tùy biến lại trong quá trình lập trình nhằm thay đổi vị trí, góc quay và tỉ lệ của đối tượng qua các đoạn mã. Từ các thành phần cơ bản này, chúng ta sẽ tạo ra Game Object với các thành phần khác, bổ sung chức năng cần thiết để

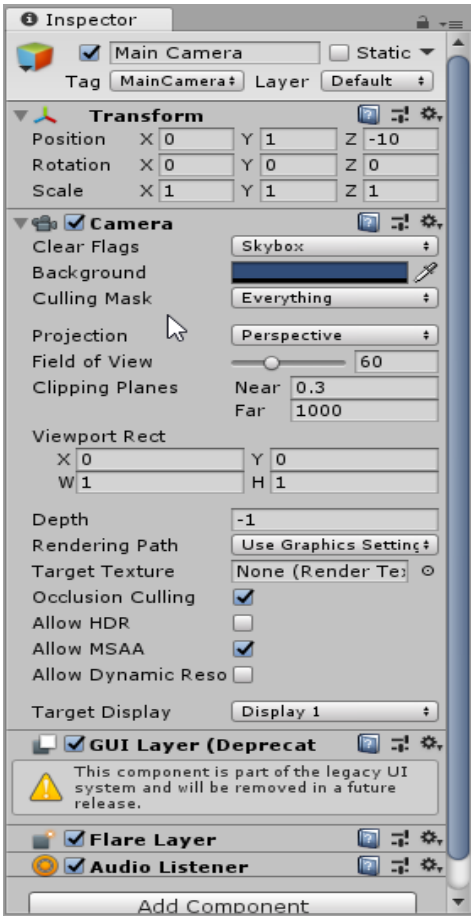
xây dựng nên bất kỳ một thành phần nào trong kịch bản Game mà chúng ta đã tưởng tượng.



Hình 4. Kéo tài nguyên vào scene để sử dụng.

2.3.4 *Components*

Components có nhiều hình thức khác nhau. Chúng có thể xác định hành vi, cách xuất hiện,… hay ảnh hưởng đến các khía cạnh khác trong chức năng của Game Object trong trò chơi. Bằng cách "gắn" chúng vào trong Game Object, chúng ta ngay lập tức có thể áp dụng tác động của chúng lên đối tượng. Những Components phổ biến trong quá trình phát triển trò chơi đều được Unity hỗ trợ sẵn. Ví dụ như thành phần Rigidbody đã được đề cập hay các yếu tố đơn giản khác như ánh sáng, Camera và nhiều thành phần khác. Để tạo nên các yếu tố tương tác trong trò chơi, chúng ta sẽ sử dụng Script (mã kịch bản), chúng cũng được xem như là một Components trong Unity.



Hình 5 Các thành phần trong đối tượng Camera.

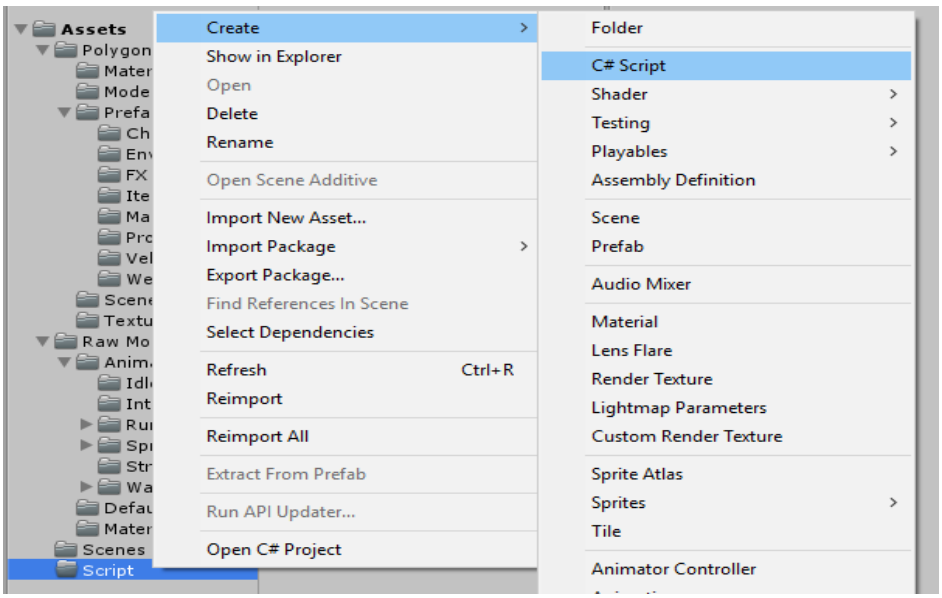
2.3.5 *Scripts*

Được Unity xem như một Components, Script là một thành phần thiết yếu trong quá trình phát triển trò chơi và đáng được đề cập đến như một khái niệm "chìa khóa". Unity cung cấp cho chúng ta khả năng viết Script bằng cả 3 loại ngôn ngữ l à: JavaScript, C# và Boo (một dẫn xuất của ngôn ngữ Python).

Unity không đòi hỏi chúng ta phải học làm thế nào để lập trình trong Unity, nhưng hầu như chúng ta phải sử dụng Script tại mỗi thành phần trong kịch bản mà chúng ta phát triển. Unity đã xây dựng sẵn một tập hợp đa dạng các lớp, hàm mà chúng ta hoàn toàn có thể ứng dụng trong quá trình lập trình cho trò chơi của mình.

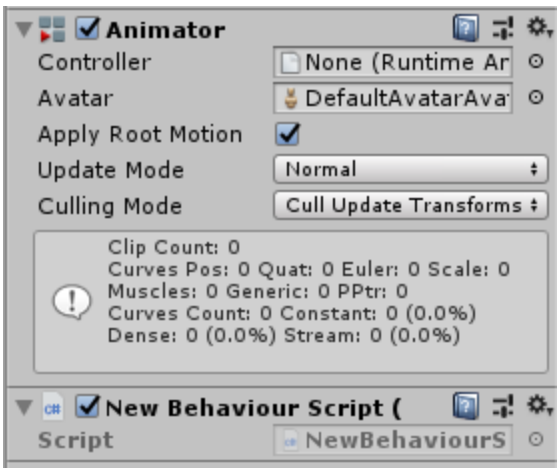
Để viết script, chúng ta sẽ làm việc với một trình biên tập Script độc lập của Unity, hoặc với chương trình Mono Developer được tích hợp và đồng bộ với Unity trong những phiên bản mới nhất hiện nay.

Mono developer là một IDE khá tốt để lập trình khi cung cấp nhiều chức năng tương tự như Visual studio. Mã nguồn viết trên Mono Developer sẽ được cập nhật và lưu trữ trong dự án Unity.



Hình 6. Cách tạo file Script mới.

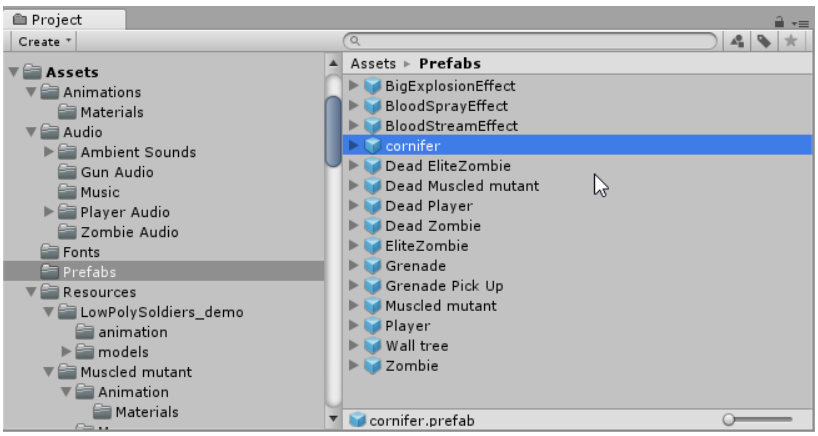
Một đoạn script muốn thực thi được thì nó phải được gắn vào một đối tượng.



Hình 7. Một file Script được gắn vào đối tượng.

2.3.6 *Prefabs*

Prefabs cho phép chúng ta lưu trữ các đối tượng với những Components và những thiết đặt hoàn chỉnh. Có thể so sánh với khái niệm cơ bản là MovieClip trong Adobe Flash, Prefabs chỉ đơn giản là một Container (một đối tượng chứa) rỗng mà chúng ta có thể đưa bất kì một đối tượng hay dữ liệu mẫu nào mà chúng ta muốn tái sử dụng về sau.

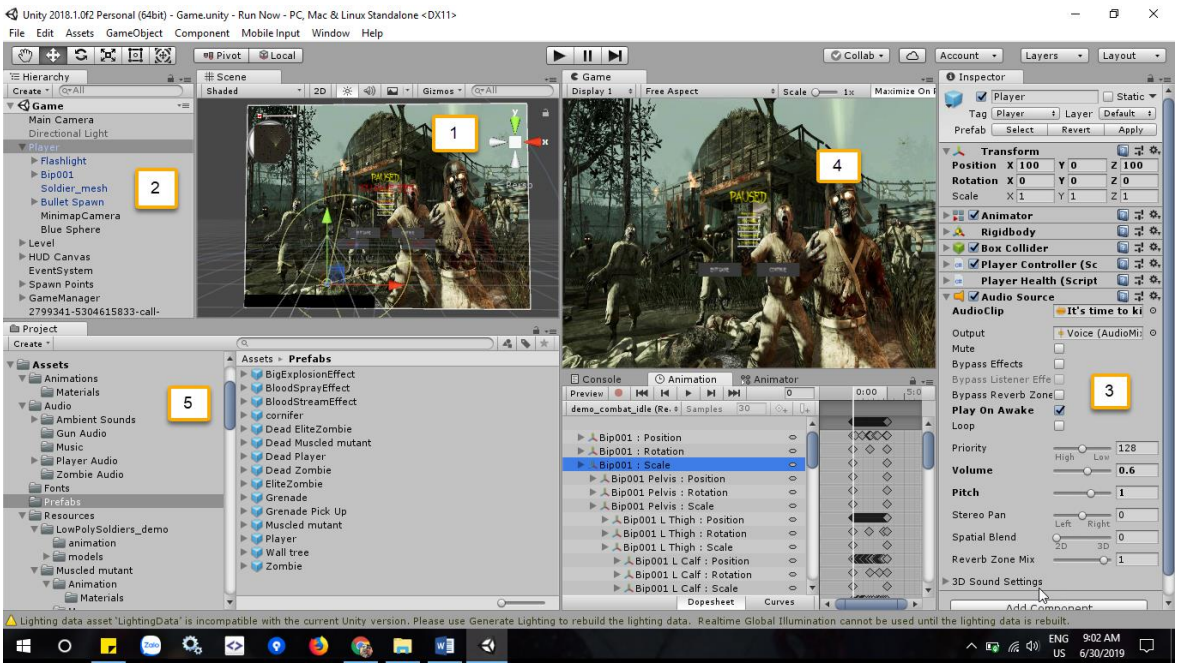


Hình 8. Một số Object trong Prefabs.

2.4 Giao diện của Unity

2.4.1 *Giao Diện*

Giao diện của Unity có khả năng tùy chỉnh bố trí tương tự như nhiều môi trường làm việc khác. Dưới đây là một kiểu bố trí điển hình trong Unity:



Hình 9. Giao diện(Layout) của Unity.

∗ Chú thích:

Scene (1): Nơi mà trò chơi sẽ được xây dựng.

Hierarchy (2): Danh sách các Game Object trong scene.

Inspector (3): Những thiết lập, thành phần, thuộc tính của đối tượng (hoặc Asset) đang được chọn.

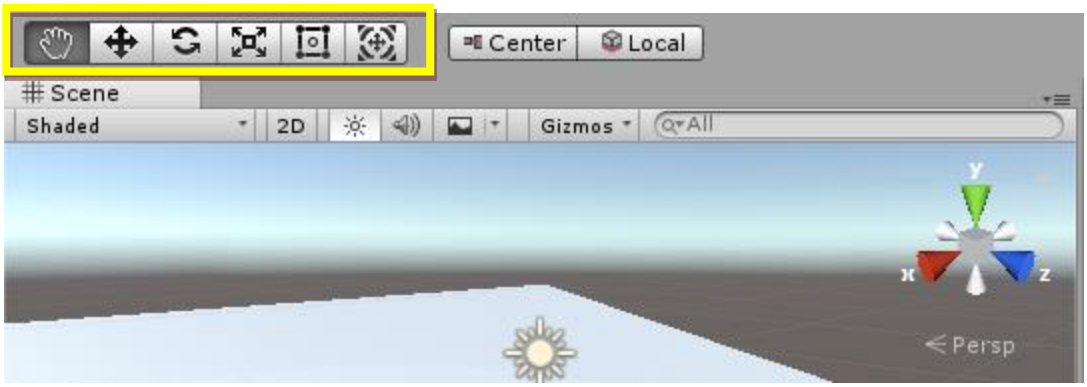
Game (4): Cửa sổ xem trước, nó chỉ hoạt động trong chế độ "Play" (Preview – xem trước).

Project (5): Danh sách các Assets của dự án, được ví như thư viện của dự án.

2.4.2 *Cửa sổ Scene và Hierarchy*

Cửa sổ scene là nơi mà chúng ta sẽ xây dựng các thực thể, đối tượng của dự án vào đó. Cửa sổ cung cấp góc nhìn phối cảnh (Perspective (góc nhìn 3D), chúng ta có thể chuyển qua các góc nhìn khác như từ trên xuống hoặc từ dưới lên (Top Down), từ trái sang phải hoặc phải sang trái (Side On), từ trước ra sau hoặc sau đến trước (Front On). Cửa sổ này sẽ kết hình xuất đầy đủ những hình ảnh trong thế giới của trò chơi mà chúng ta tạo ra dưới dạng một vùng biên tập mà chúng ta có thể biên tập, chỉnh sửa trực tiếp thế giới đó.

Khi kéo thả Assest vào cửa sổ Scene, Assets sẽ trở thành Game Object. Cửa sổ Scene được ràng buộc cùng với cửa sổ Hierarchy, cửa sổ Hierarchy liệt kệ danh sách các Game Object có trong Scene và được sắp xếp theo thứ tự chữ cái từ A-Z.



Hình 10. Các nút chức năng,trong cửa sổ scene.

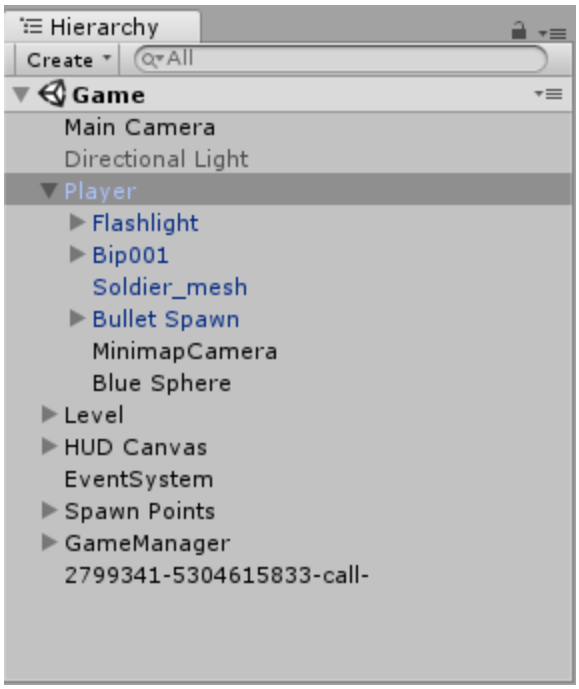
Cửa sổ Scene còn đi kèm với 4 bốn nút chức năng hữu ích được hiển thị dưới dạng hình ảnh như trên. Chúng có thể được lựa chọn thông qua các phím tắt Q, W, E và R. Những nút này có các chức năng như sau:

Công cụ bàn tay (Q): Công cụ này cho phép chúng ta di chuyển đến một khu vực nào đó trong Scene bằng thao tác kéo thả thuộc trái.

Công cụ di chuyển (W): Công cụ này cho phép chúng ta chọn một đối tượng trong cảnh và thực hiện thao tác di chuyển, thay đổi vị trí của đối tượng đó. Khi chọn, tại vị trí của đối tượng sẽ hiển thị các trục và mặt phẳng gắn liền với đối tượng cho phép chúng ta di chuyển đối tượng trượt theo các trục, mặt phẳng hoặc di chuyển một cách tùy ý.

Công cụ xoay (E): Công cụ này có đặc điểm và cách sử dụng giống với công cụ di chuyển, tuy nhiên thay vì để di chuyển vị trí của đối tượng thì công cụ này giúp chúng ta xoay đối tượng xoay quanh trục hay tâm của đối tượng.

Công cụ điều chỉnh tỉ lệ (R): Cũng tương tự như công cụ di chuyển và xoay, công cụ này cho phép chúng ta tùy chỉnh kích thước, tỉ lệ của đối tượng một cách tùy ý



Hình 11. Cửa sổ Hierarchy.

2.4.3 *Cửa sổ Inspector*

Cửa sổ Inspector có thể được xem như một công cụ cho phép chúng ta tùy chỉnh các thiết đặt, các thành phần của Game Object hoặc Assets đang được chọn.

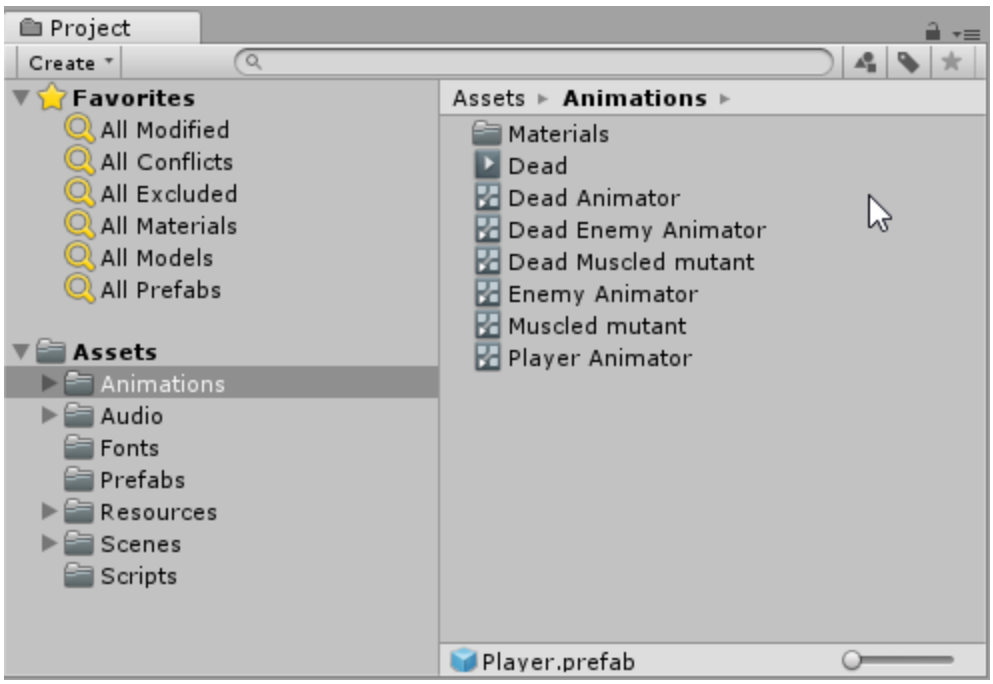
Cửa sổ này sẽ hiển thị đầy đủ các Components của đối tượng mà chúng ta chọn. Nó cho phép chúng ta điều chỉnh các biến của Components dưới các hình thức như: Textbox, Slider, Button, Drop-dowm Menu…

Ngoài việc hiển thị các Component của đối tượng được chọn, cửa sổ Inspector còn hiển thị các thiết đặt chung của hệ thống hay của trò chơi khi ta chọn chúng từ menu Edit.

Hình 12. Cửa sổ Inspector.

Trong hình trên, chúng ta thấy cửa sổ Inspector đang hiển thị một vài thuộc tính, Components của một đối tượng đang được chọn. Trong đó, bao gồm 2 Components là Transform và Animation. Cửa sổ Inspector sẽ cho phép chúng ta thay đổi các thiết đặt trên. Các Components này còn có thể được tạm thời vô hiệu hóa vào bất kỳ lúc nào chúng ta muốn bằng cách bỏ chọn Checkbox ở góc trên bên trái của mỗi Component, việc này sẽ rất hữu ích cho chúng ta khi muốn kiểm tra hay thử nghiệm các Components này. Ngoài ra, cửa Inspector còn cho phép chúng ta vô hiệu hóa toàn bộ một đối tượng đang được chọn bằng cách bỏ chọn Checkbox ở trên cùng góc trái của cửa sổ Inspector.

2.4.4 *Cửa sổ Project*



Hình 13. Cửa sổ Project.

Cửa sổ Project là cửa sổ cho phép chúng ta nhìn thấy trực tiếp nội dung của thư mục Assets của dự án. Mỗi dự án Unity đều được chứa trong một thư mục cha. Trong đó có 3 thư mục con là Assets, Library và Temp (chỉ có khi Unity đang chạy). Đặt tất cả các Assets vào thư mục Assets có nghĩa là ngay lập tức chúng ta sẽ

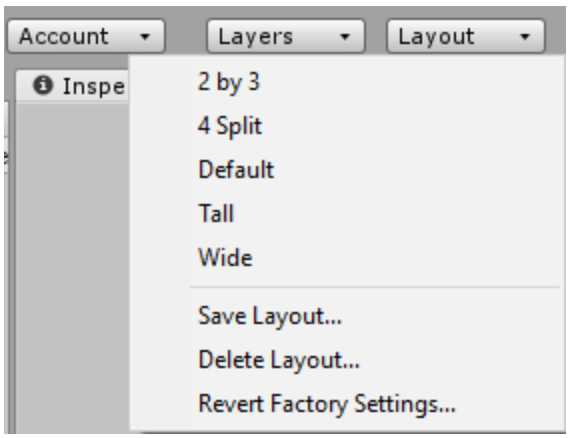
thấy chúng xuất hiện trong cửa sổ Project. Ngoài ra, khi thay đổi vị trí của Assets trong thư mục Assets hay lưu tập tin lại từ một chương trình ứng dụng thứ 3 nào khác (ví dụ như Photoshop), sẽ làm cho Unity nhập lại (Re-Import) Assets, phản ánh sự thay đổi này ngay lập tức trong cửa sổ Project và Scene có sử dụng Assets vừa được thay đổi.

Cửa sổ Project được tích hợp nút Create, nút này cho phép chúng ta tạo mới bất kì một Assets mới nào, ví dụ như Script, Prefabs, Materials, …

2.4.5 *Cửa sổ Game*

Cửa sổ Game sẽ được gọi khi chúng ta nhấn vào nút Play (là một hành động thực hiện test trò chơi). Cửa sổ này cho phép chúng ta tùy chọn về thiết đặt tỉ lệ màn hình, nó phản ánh phạm vi trong Scene mà người chơi có thể thấy được với mỗi tỉ lệ màn hình tương ứng, ví dụ như với mỗi tỉ lệ màn hình 4:3, 16:9 thì người chơi sẽ có một phạm vi nhìn thấy khác nhau.

Sau khi nhấn vào nút Play, chúng ta sẽ ở chế độ Testing, lúc này mọi thay đổi về các thuộc tính, Components,… của đối tượng sẽ chỉ là tạm thời. Tức là chúng sẽ trở về như ban đầu (trước khi nhấn nút Play) sau khi kết thúc chế độ Testing.



Hình 14. Các loại hình ảnh trong cửa sổ game.

Chương 3 . MỘT SỐ VẤN ĐỀ VÀ GIẢI PHÁP

3.1 Chuyển động mô hình nhân vật 3D

3.1.1 *Vấn Đề*

Chúng ta đã load được mô hình 2D,3D vào trong game, vậy làm sao để mô hình 2D,3D này có thể chuyển động trong game.

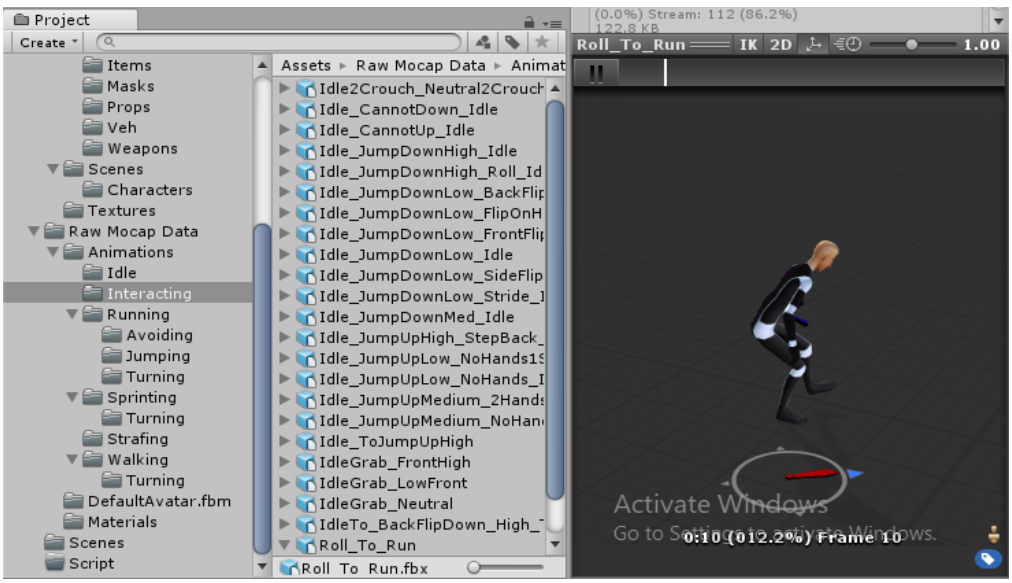
3.1.2 *Giải Pháp*

Trước tiên mô hình 2D,3D cần phải có sẵn animation bên trong. Khi import mô hình vào Unity, animation trong mô hình được tự động chuyển thành một AnimationClip. Điều này giúp animation này có thể dùng cho các mô hình khác trong project.

Trước hết ta phải tạo AnimationClip từ animation có sẵn của mô hình.

Có 2 loại mô hình 2D có sẵn animation:

Loại thứ nhất:

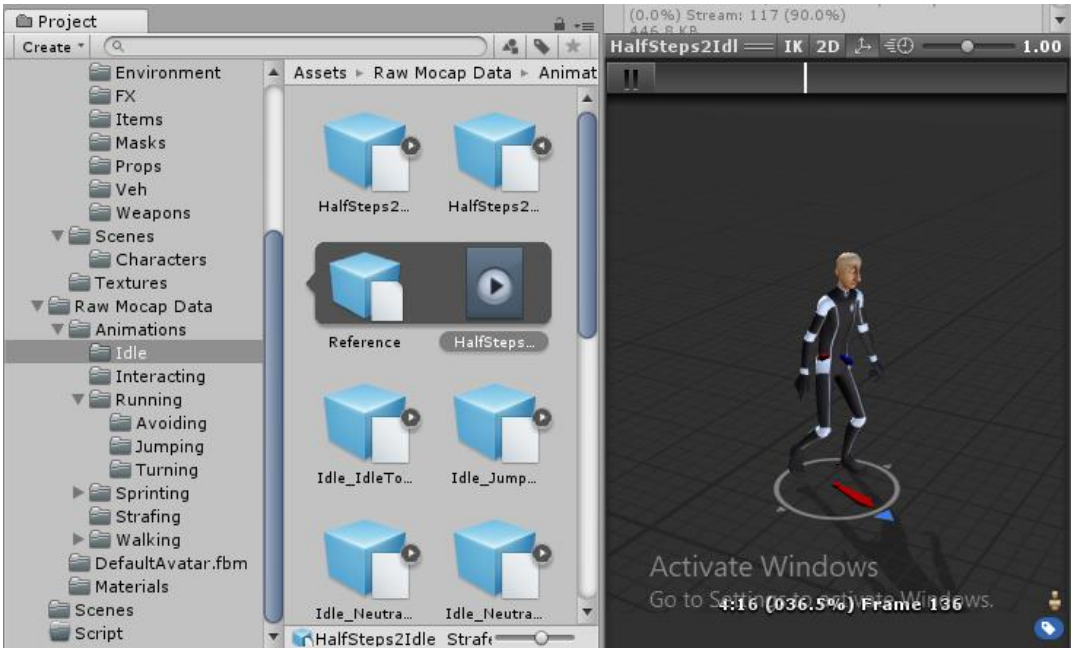


Hình 15. Mô hình 2D,3D bên trong chứa nhiều Animation.

Mô hình 2D, 3D trên sau khi import vào project game, bên trong đã có sẵn 9 Animation, mỗi Animation tự động được tạo thành một AnimationClip bên sẽ trong đối tượng game.

Loại thứ hai:

Mô hình 2D,3D chỉ chứa một Animation.



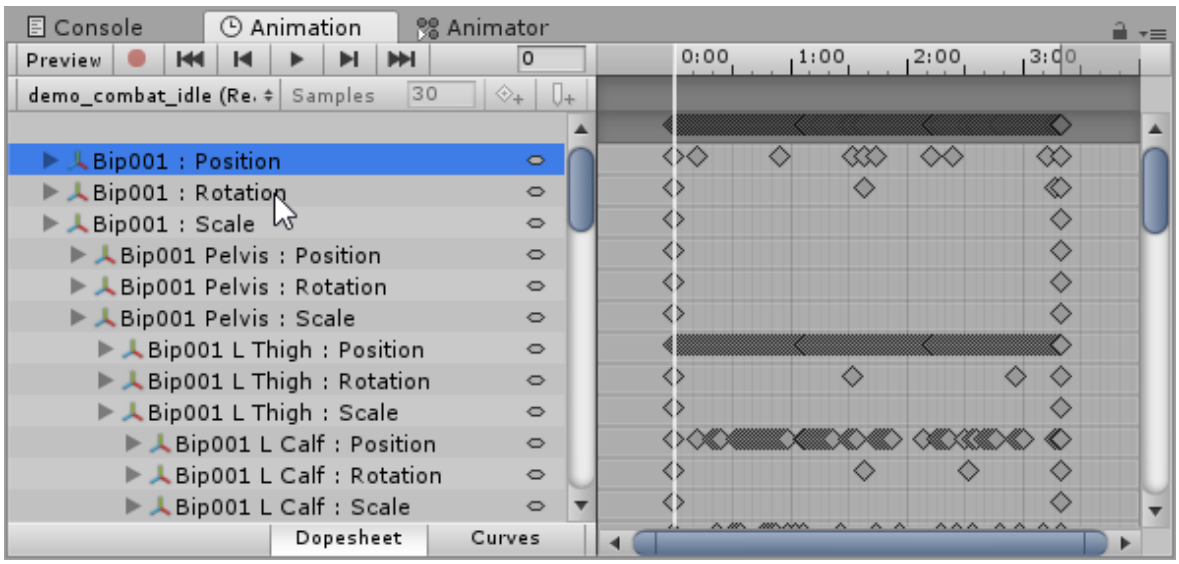
Hình 16. Mô hình 2D, 3D chứa một Animation.

Mô hình trên, mỗi mô hình sau khi import có một AnimationClip duy nhất có tên "Default Take". Về bản chất hình dạng mô hình là như nhau, chỉ khác nhau Animation (Idle, Run, Walk). Vậy làm sao chúng ta kết hợp các AnimationClip này vào đối tượng game duy nhất.Unity quy định như sau:

Lấy một mô hình làm mô hình chính,có thế không cân Animation kèm theo cũng được.

Các mô hình còn lại, tên phải có 2 phần cách nhau bởi '@', phần đầu phải trùng tên với mô hình chính đã chọn, phần thứ 2 sẽ là tên Animation.

Với cách đặt tên như vậy, khi đưa các mô hình này vào project để sử dụng thì Unity sẽ tự động dổi tên Animation mặc định trong mô hình thành tên trùng với phần tên mô hình nằm sau chữ '@'. Lưu ý là phải đổi tên cho mô hình từ bên ngoài project thì tên của Animation của mô hình đó sẽ không bị thay đổi theo phần tên sau dâu '@'.



Hình 17. Animation.

Tóm Lại :

Việc gọi thực hiện các animation của đối tượng là khá đơn giản. Tuy nhiên phải quyết định chọn mô hình loại nào để có thể thêm hoặc bớt animation cho mô hình dễ dàng. Nếu chọn mô hình loại 1 thì chúng ta phải import vào các chương trình hỗ trợ làm animation cho mô hình để chỉnh sửa thêm xóa animation rồi import vào Unity lại, còn chọn mô hình loại 2 thì chúng ta chỉ cần xóa hay thêm file mô hình là xong, rất linh hoạt và nhanh chóng.

3.2 Xây dựng giao diện game

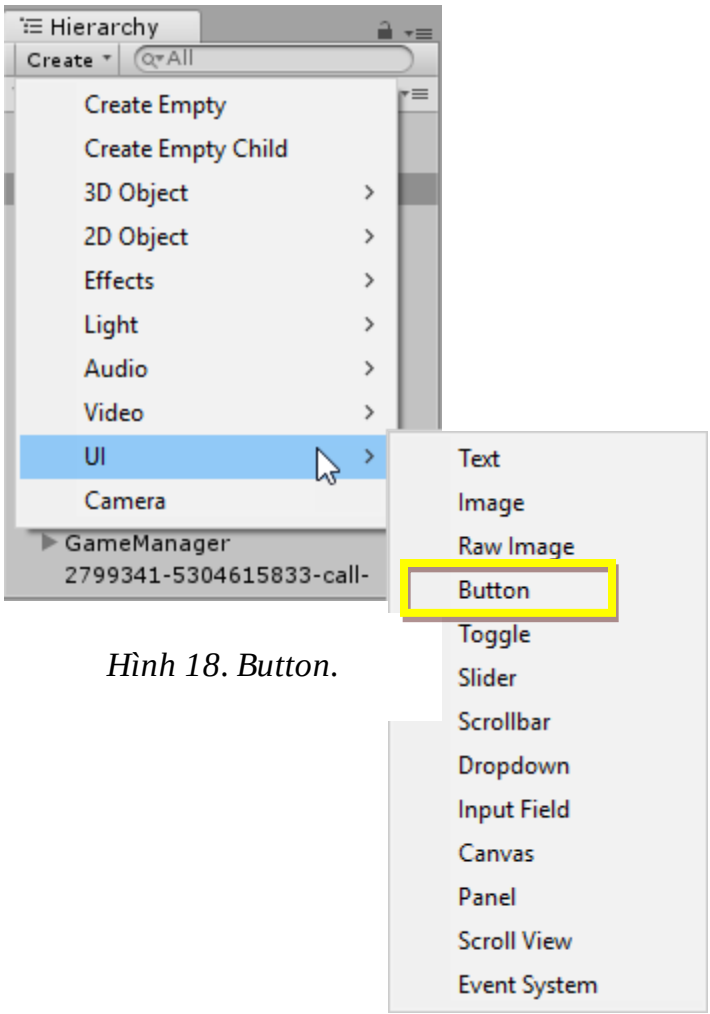
3.2.1 *Vấn đề*

Giao diện đồ họa người dùng là một phần quan trọng không thể thiếu trong khi xây dựng một ứng dụng game hay bất cứ một ứng dụng

nào để vẽ các đối tượng đồ họa như Button, Label, Checkbox, Slider, … lên màn hình.

3.2.2 *Giải pháp*

Để làm được điều này chúng ta dùng lớp GUI, GUI là chữ viết tắ t của "Graphical User Interface" – "Giao diện đồ họ a người dùng". Hệ thống GUI của Unity được gọi là GUIUnity. Để sử dụng được các phương thức trong GUI ta phải gọi thực hiện từ trong hàm OnGUI() giống như sự kiện Paint trong C#. Ví dụ sau đây sẽ tạo ra một button đơn giản:



Hình 18. Button.

Tóm Lại :

Với lớp GUI trong Unity, chúng ta hoàn toàn có xây dựng nên một giao diện tuyệt vời cho ứng dụng game. Ngoài các phương thức của lớp GUI đã nêu trên thì còn rất nhiều phương thức vẽ các đối tượng khác như Radio, Checkbox, Slider…

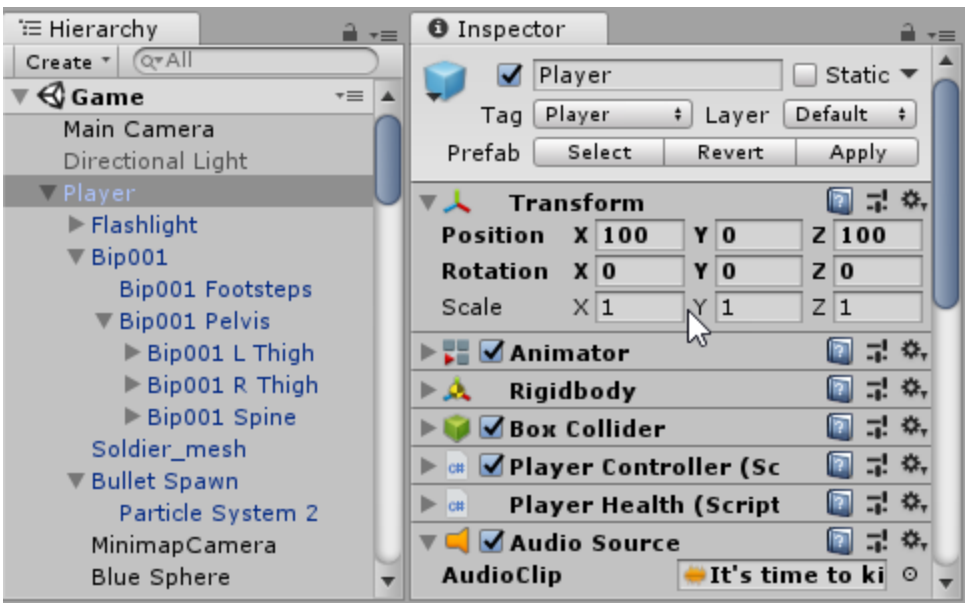
3.3 Âm thanh trong game

3.3.1 *Vấn đề*

Âm thanh là yếu tố không kém phần quan trọng trong ứng dụng game. Thật nhàm chán khi một cảnh đánh đánh nhau, bắn nhau hay các hiệu ứng đẹp mắt mà không có âm thanh. Âm thanh 3 chiều sẽ làm cho game thực hơn và sống động hơn.

3.3.2 *Giải pháp*

Để chơi được một file âm thanh trong Unity có 2 cách: bằng code hoặc trên giao diện. Dù chọn cách nào thì trước hết chúng ta phải có sẵn các file âm thanh và import vào project. Sau khi import âm thanh vào project, nếu file hợp lệ chúng ta sẽ thấy như hình sau và có thể nhấn nút play để nghe thử.

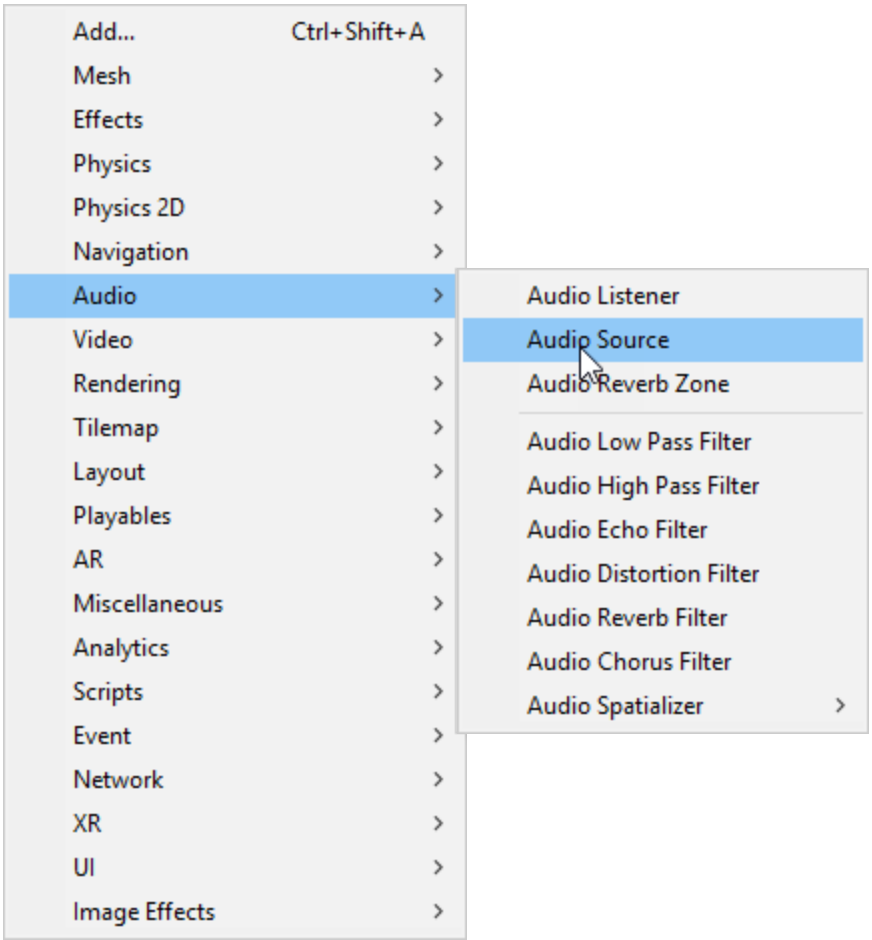


Hình 19. Thông tin âm thanh.

Cách 1: Tạo trên giao diện

Trên menu của Unity, vào GameObject => Create Empty.

Chọn đối tượng vừa tạo và gắn thành phần "AudioSource" cho đối tượng này. AudioSource là một đối tượng âm thanh. Muốn Play hay Stop, thay đổi cách lặp, tăng giảm volume nhạc thì phải thông qua đối tượng này.



Hình 20. Thêm thành Audio Source.

Sau khi gắn thành phần âm thanh cho đối tượng vừa tạo, chúng ta dễ dàng chỉnh sửa các thông số và gán file âm thanh cho thành phần AudioSource này.

Việc tạo đối tượng âm thanh trên giao diện khá đơn giản, nhưng để áp dụng vào cho game thì không được linh hoạt bằng cách sử dụng script.

Cách 2: Cách chơi nhạc bằng code

Trước tiên chúng ta cần import file âm thanh vào project trước.

Khởi tạo đối tượng game âm thanh AudioSource như sau:

Gán đường dẫn file nhạc cho âm thanh:

Sau đó chỉ cần gọi các phương thức Play() để chạy file âm thanh:

Tóm Lại :

Để chơi được âm thanh trong Unity thì chỉ cần áp dụng các kỹ thuật nêu trên là đủ. Ngoài ra còn nhiều thành phần khác như: AudioListener, AudioSetting để tạo hiệu ứng âm thanh 3 chiều thực hơn cho game.

Chương 4 . ỨNG DỤNG GAME PHÁT TRIỂN TRÊN

4.1 Giới thiệu game

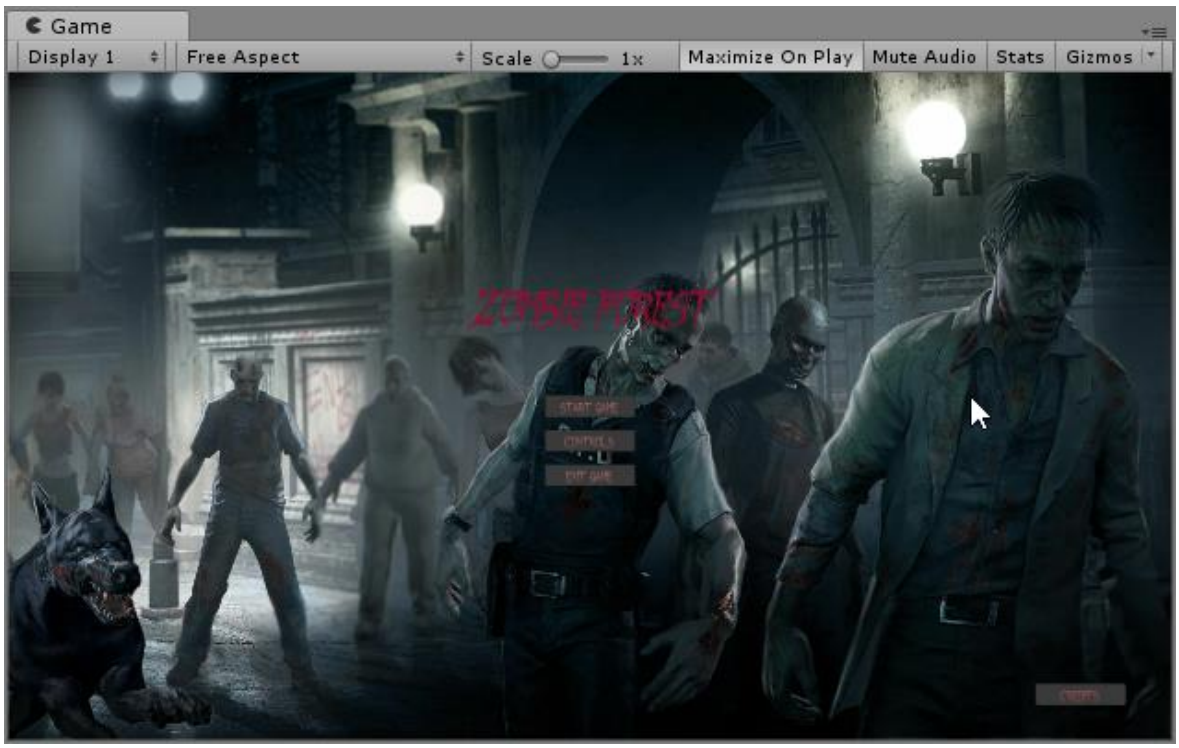
Tên game : Run Now

Đặc trưng của dòng game theo chế độ sinh tồn mà em đã xây dựng tại 1 thời điểm chỉ có một người chơi, trò chơi sẽ có 1 màn hình chơi, người chơi sẽ được cung cấp 1 lượng máu và vũ khí nhất định (súng và bom), người chơi sẽ bắn để loại bỏ quân địch để được sống sót, càng loại bỏ được nhiều quân địch thì độ khó sẽ tăng lên, khi người chơi bị quân địch gây sát thương quá nhiều dẫn đến hết máu thì người chơi sẽ chết và màn hình GameOver sẽ xuất hiên thông báo cho người chơi số điểm.

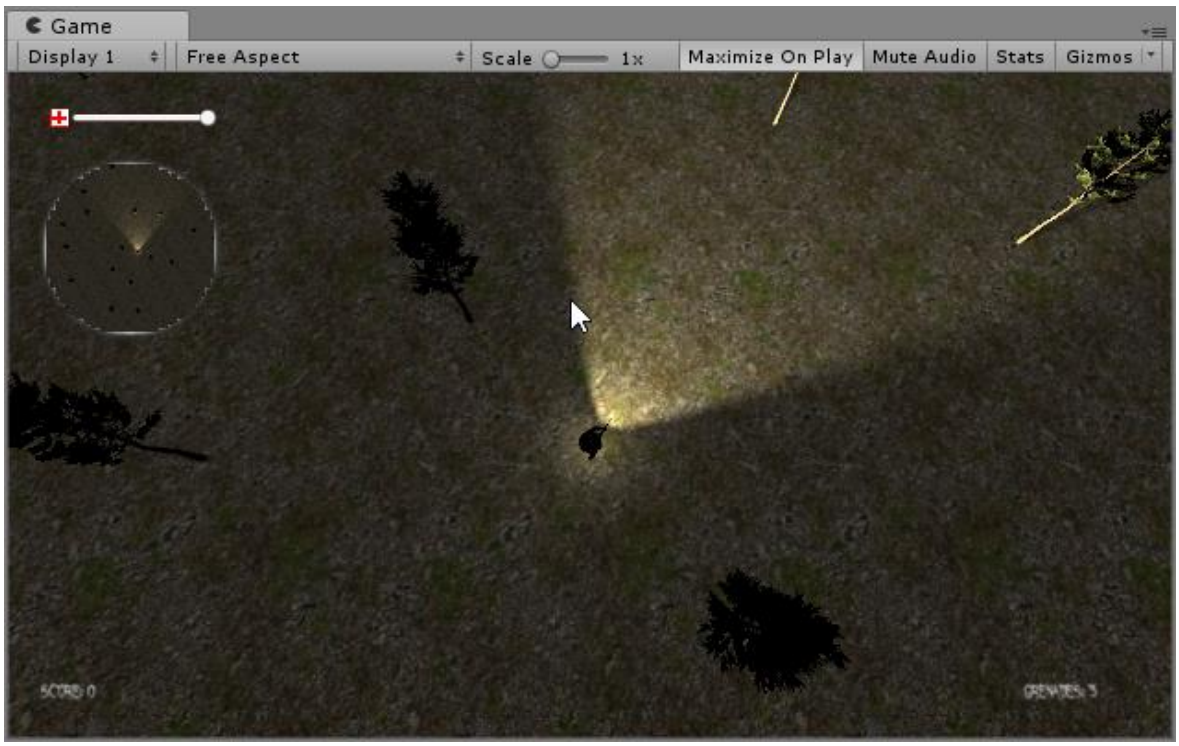
Map tối có đèn soi sáng chỉ đủ 1 góc tâm nhìn,với âm thanh thể hiện lên sợ hại làm cho người chơi them phần hồi hộp.

Màn hình sẽ di chuyển theo nhân vật và quân địch sẽ tự sinh ra trong map rồi tấn công nhân vật, càng về sau quân địch càng đông vì vậy người chơi phải khéo léo di chuyển để tiêu diệt kẻ thù trước khi kẻ thù tấn công mình nhiều để nâng cao số điểm, trên map có vật phẩm (bom) mà người có thế nhặt để sử dụng.

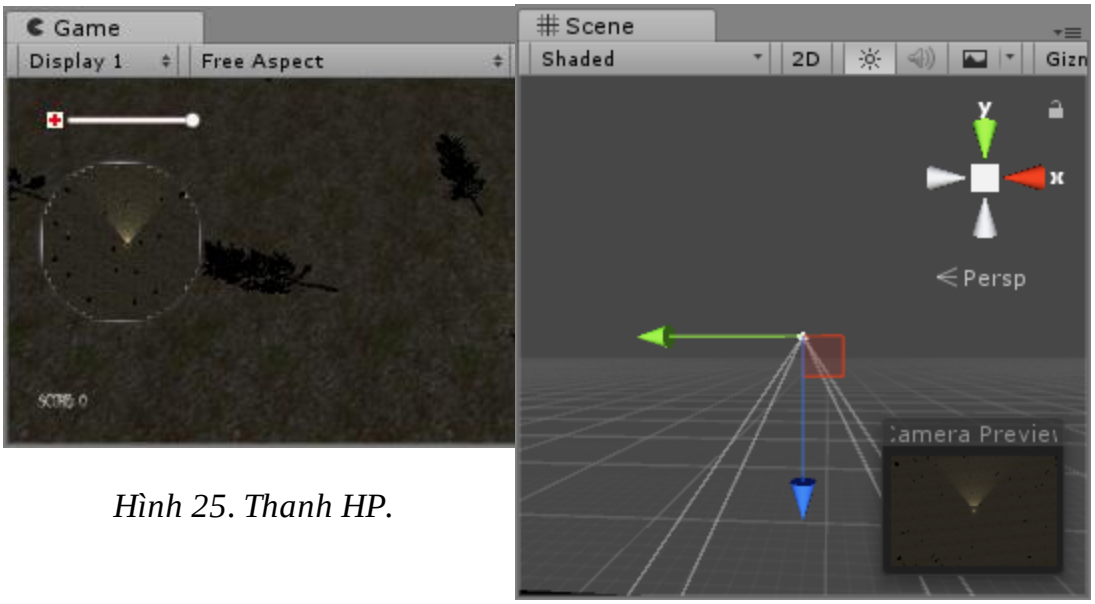
Nếu người chơi có thể dùng ESC để tạm dừng trận đấu, có button chỉnh sửa âm thanh nhạc và game chỉ kết thúc khi người chơi hết máu sau khi Game Over thì sẽ hiện ra giao diện button để bạn có thể chọn chức restart.



Hình 22. Màn hình Controls.

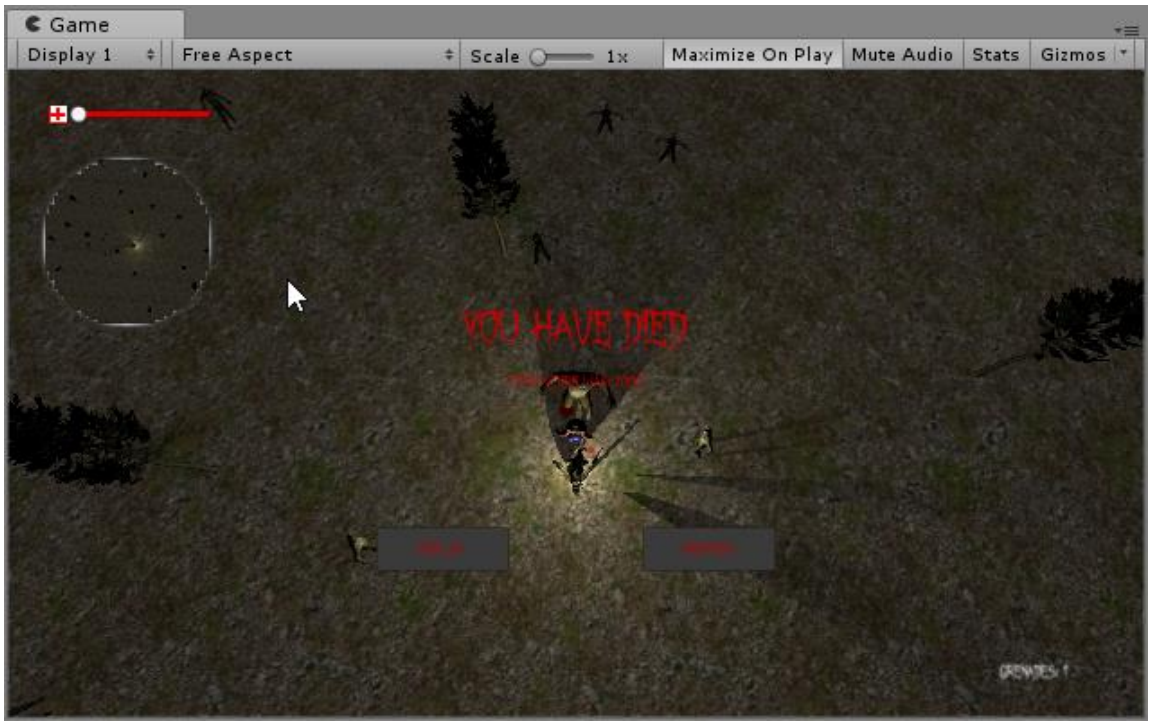


Hình 21. Màn hình chính.

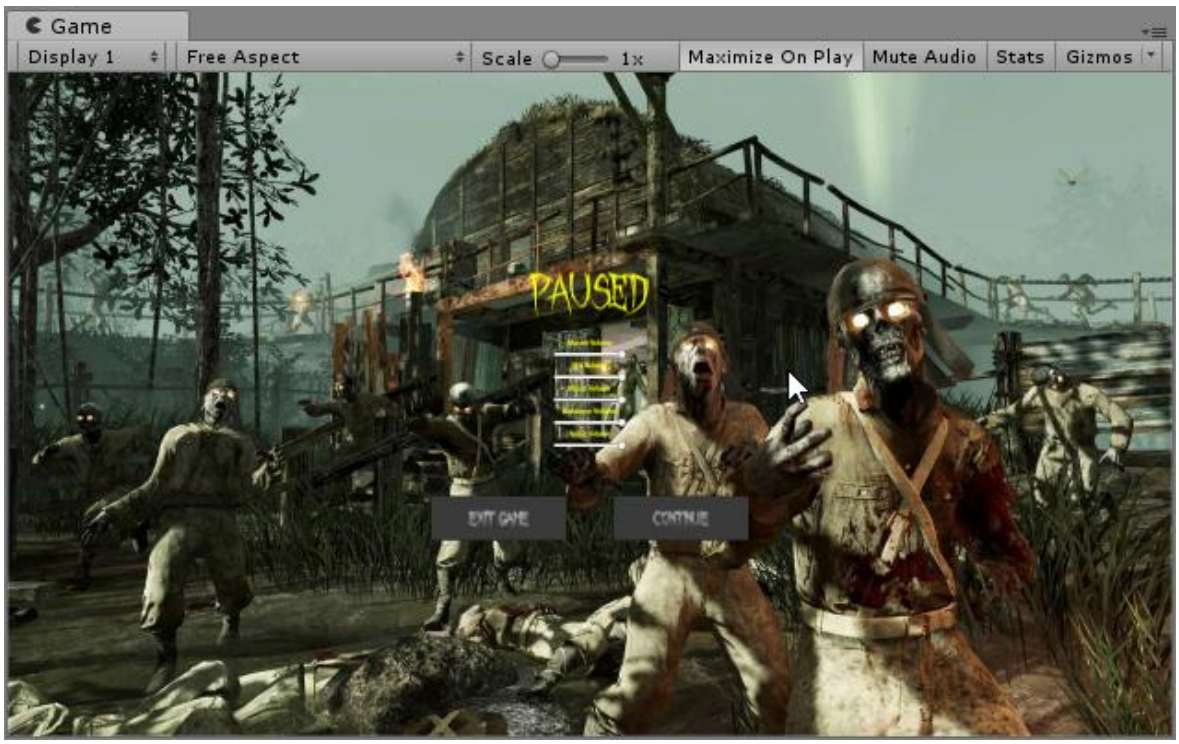


Hình 25. Thanh HP.

Hình 24. Minimap.



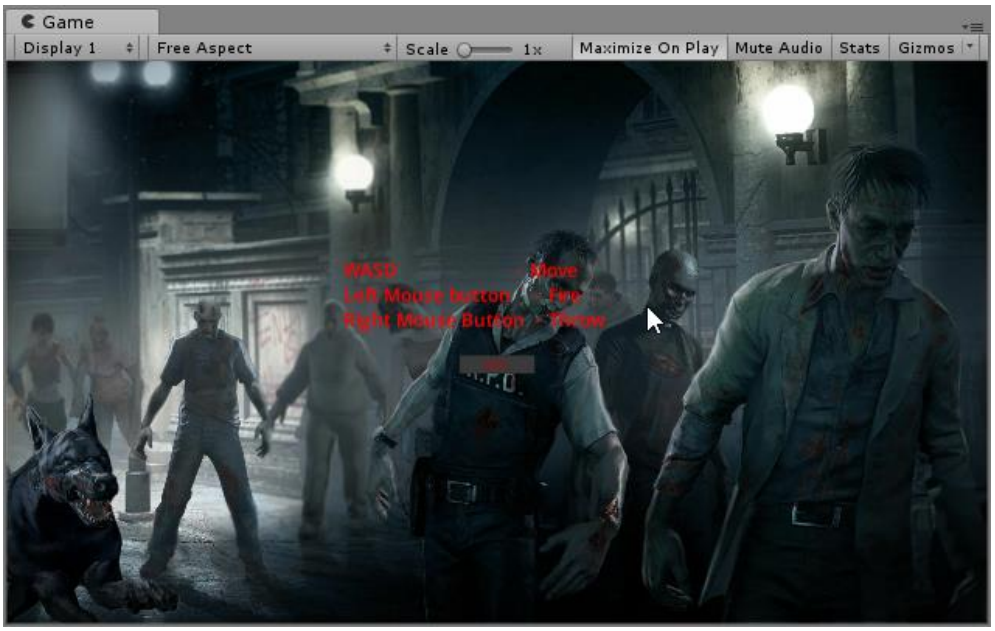
Hình 23. Màn hình lúc chết.



Hình 26. Màn hình Paused (ESC)

4.2 Các quy luật chơi chính

4.2.1 *Di chuyển*



Hình 27. Màn hình Controls.

Người chơi sẽ di chuyển bằng các nút trên bàn phím, di chuyển qua trái (← hoặc A), qua phải (→ hoặc D), lên (↑ hoặc W), xuống (↓ hoặc S) để né kẻ thù và tiêu diệt kẻ thù, làm sao chơi được càng lâu thì càng được nhiều điểm, người chơi chỉ có thể di chuyển trong phạm vi map.

Hình 28. Item Bom.

Dùng chuột để điều khiển tâm nhìn camera, click button right thì sẽ bắn đạn, còn click button left thì sẽ nem bom (với lục ném dài tới 3m). Nên di chuyển để tìm bom và lấy chúng.

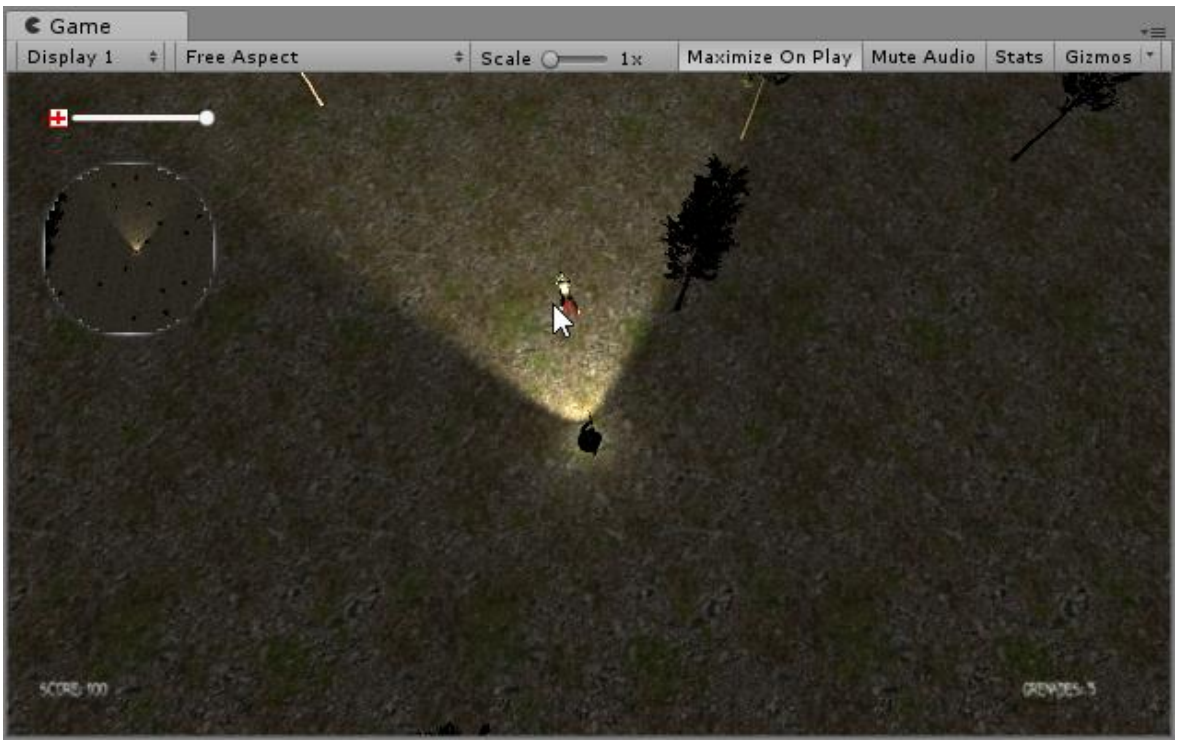
4.2.2 *Tấn công*

Người chơi sẽ dùng súng bắn đạn (đạn không giới hạn) hoặc bom để tiêu diệt kẻ thù, mỗi kẻ thù sẽ có một lượng máu nhất định :

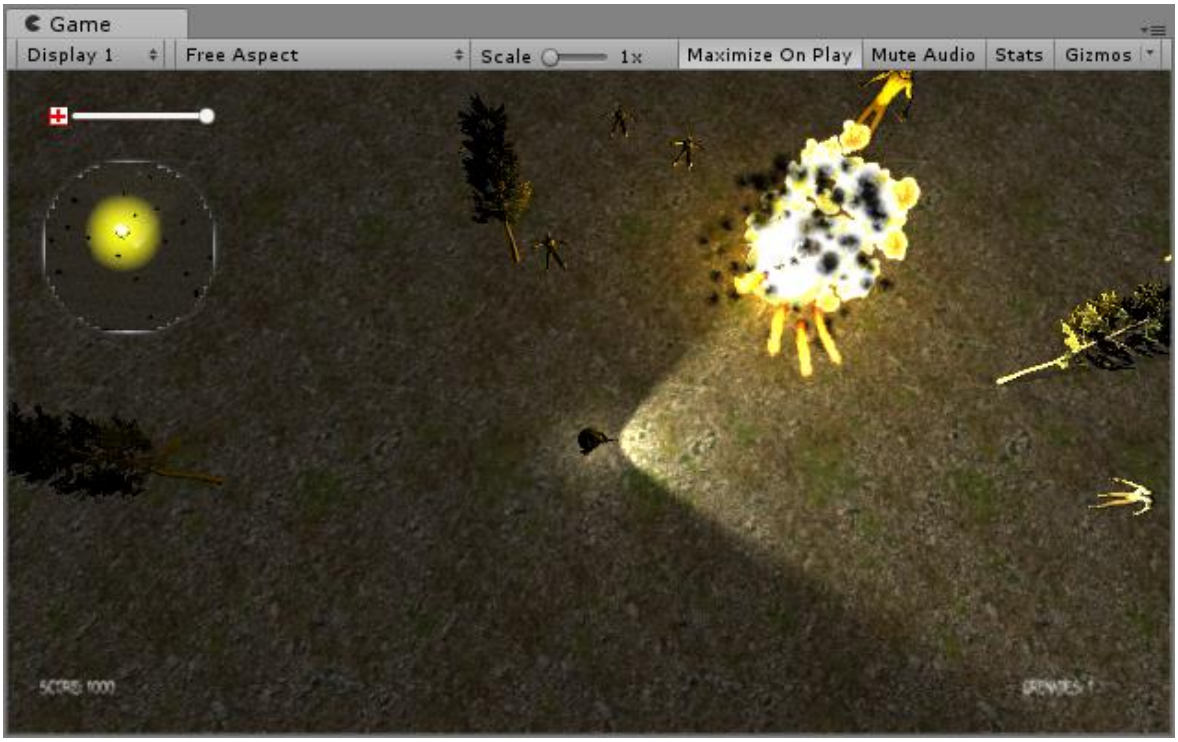
Zombie Normal

Zombie Speed : Máu it,tốc độ nhanh.

Zombie Tanker : Nhiều máu,Khổng lồ,tốc độ chậm.Nếu để kẻ thù đánh trúng thì mỗi lần sẽ bị trừ đi 20 máu trên tổng số 100.



Hình 29. Giết Zombie.



Hình 30. Kích hoạt Bom.

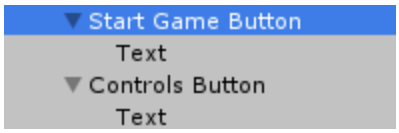
Chương 5 . DỮ LIỆU GAME RUN NOW

5.1 Màn hình bắt đầu lập trình

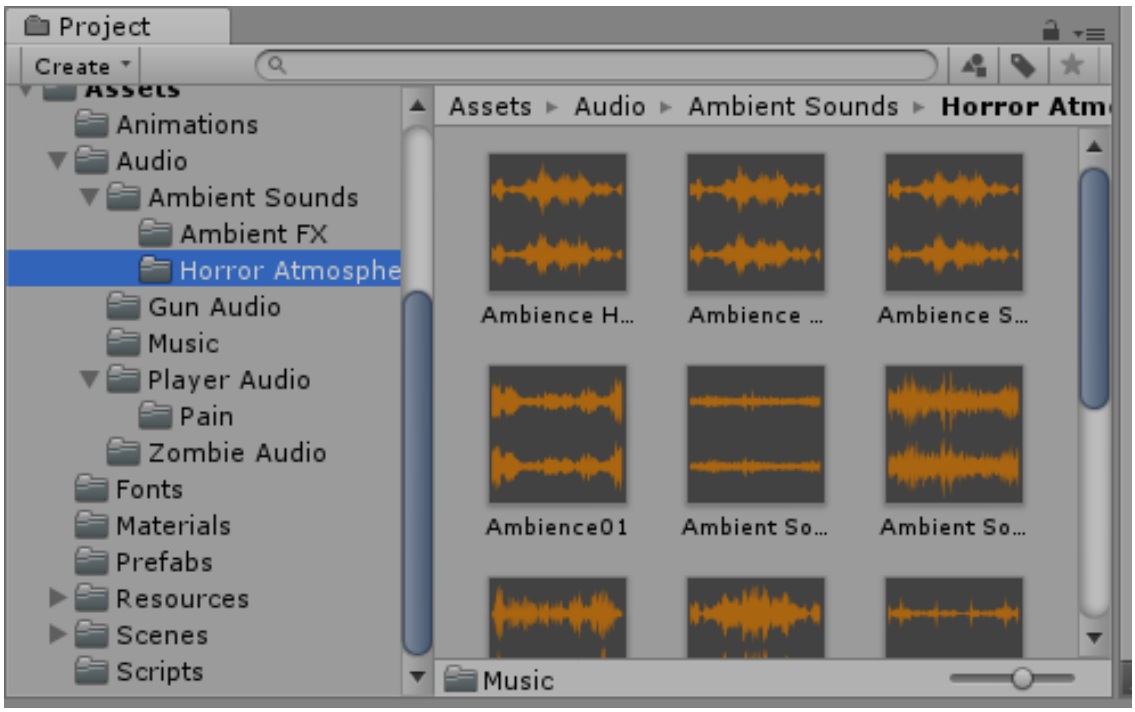
Game được hình thành bởi 2 Sence trên màn hình và nhiều Animation and Audio,Button,Text..

Hình 34. Scene trong 'Run Now'.

Hình 33. Animations trong 'Run Now'.



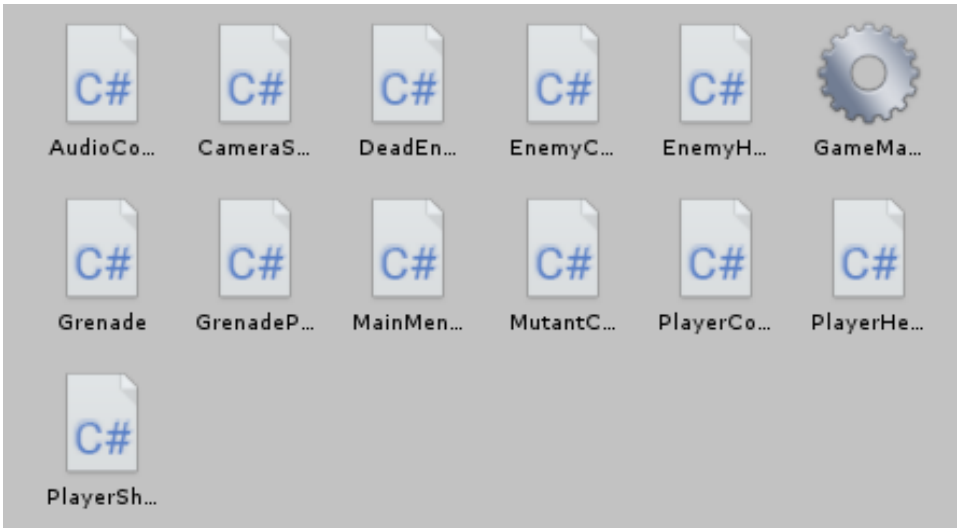
Hình 32. Button and Text trong 'Run Now'.



Hình 31. Audio trong 'Run Now'.

5.2 Code trong game

Ngôn ngữ em chọn để lập trình game Run Now là ngôn ngữ C# trên MonoDeveloper được Unity hỗ trợ



Hình 35. Button and Text trong 'Run Now'.

5.2.1 *CameraScript*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class CameraScript : MonoBehaviour
{
    public Transform target;
// Vị trí mà máy ảnh đó sẽ theo dõi mục tiêu khi chuyển động.
    public float smoothing = 5f; // Tốc độ máy ảnh sẽ theo dõi
    Vector3 offset;         // Giá trị bù ban đầu từ mục tiêu.
    void Start ()
```

```
    {
        // Tính toán bù đắp ban đầu.
        offset = transform.position - target.position;
    }
    void FixedUpdate ()
    {
        // Tạo một postion camera đang nhắm đến dựa trên offset từ mục tiêu.
        Vector3 targetCamPos = target.position + offset;
        // Nội suy mượt mà giữa vị trí hiện tại của máy ảnh và vị trí mục tiêu của nó.
        transform.position = Vector3.Lerp (transform.position, targetCamPos, smoothi
ng * Time.deltaTime);
    }
}
```

5.2.2 *PlayHealth*

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
public class PlayerHealth : MonoBehaviour {
    public const int maxHealth = 100;
    public int currentHealth = maxHealth;
    public Slider healthSlider;
    public GameObject deadPlayerPrefab;
    PlayerController playerController;
    PlayerShoot playerShoot;
    Animator anim;
    AudioSource playerAudio;
```

```
    AudioSource heartBeat;
    AudioSource[] AmbientSounds;
    GameManager gameManager;
    public AudioClip pain1Clip;
    public AudioClip pain2Clip;
    public AudioClip pain3Clip;
    public AudioClip pain4Clip;
    public AudioClip pain5Clip;
    public AudioClip pain6Clip;
    public AudioClip pain7Clip;
    public AudioClip pain8Clip;
    AudioClip[] painClips;
    int painClipsIndex = 0;
    public AudioClip deathClip;
    public AudioClip heartBeatSlow;
    public AudioClip heartBeatFast;
    void Awake ()
    {
        gameManager = GameObject.FindGameObjectWithTag ("GameController").G
etComponent<GameManager> ();
        AmbientSounds = GameObject.FindGameObjectWithTag ("GameController").
GetComponents<AudioSource> ();
        anim = GetComponent<Animator> ();
        playerAudio = GetComponent <AudioSource> ();
        heartBeat = AmbientSounds [1];
        playerController = GetComponent <PlayerController> ();
        playerShoot = GetComponentInChildren <PlayerShoot> ();
        painClips = new AudioClip[8];
        painClips.SetValue (pain1Clip, 0);
```

```
    painClips.SetValue (pain2Clip, 1);
    painClips.SetValue (pain3Clip, 2);
    painClips.SetValue (pain4Clip, 3);
    painClips.SetValue (pain5Clip, 4);
    painClips.SetValue (pain6Clip, 5);
    painClips.SetValue (pain7Clip, 6);
    painClips.SetValue (pain8Clip, 7);
}
public void TakeDamage(int amount)
{
    playerAudio.clip = painClips [painClipsIndex];
    playerAudio.Play ();
    painClipsIndex += 1;
    if (painClipsIndex >= 8) {
        painClipsIndex = 0;
    }
    currentHealth -= amount;
    healthSlider.value = currentHealth;
    if (currentHealth > 20) {
        heartBeat.Pause ();
    }
    if ((currentHealth <= 20) && (currentHealth > 10)){
        heartBeat.clip = heartBeatSlow;
        heartBeat.Play ();
    }
    if ((currentHealth <= 10) && (currentHealth > 0))
    {
        heartBeat.clip = heartBeatFast;
        heartBeat.Play ();
```

```
        }
        if (currentHealth <= 0)
        {
            Die ();
        }
    }
    void Die()
    {
        anim.SetBool ("IsDead", true);
        playerController.enabled = false;
        playerShoot.enabled = false;
        playerAudio.clip = deathClip;
        playerAudio.Play ();
        heartBeat.enabled = false;
        GameObject flashlight = GameObject.FindGameObjectWithTag ("Light");
       GameObject meshrenderer = GameObject.FindGameObjectWithTag("Rendere
r");
        flashlight.SetActive (false);
        meshrenderer.SetActive (false);
        Instantiate (deadPlayerPrefab, transform.position, transform.rotation);
        gameManager.GameOver ();
    }}
```

5.2.3 *PlayController*

```
using System.Collections;
using System.Collections.Generic;
using UnityStandardAssets.CrossPlatformInput;
using UnityEngine;
public class PlayerController : MonoBehaviour {
```

```
public Transform bulletSpawn;
public float speed = 4f;
float camRayLength  = 100f;
Vector3 movement;
Rigidbody playerRigidbody;
int floorMask;
Animator anim;
void Awake ()
{
    // Create a layer mask for the floor layer.
    floorMask = LayerMask.GetMask ("Floor");
    // Set up references.
    anim = GetComponent <Animator> ();
    playerRigidbody = GetComponent <Rigidbody> ();
}
void FixedUpdate()
{
 float h = CrossPlatformInputManager.GetAxisRaw("Horizontal");
 float v = CrossPlatformInputManager.GetAxisRaw("Vertical");
    Move (h, v);
    Turning ();
    Animating (h, v);
}
void Move (float h, float v)
{
    // Set the movement vector based on the axis input.
    movement.Set (h, 0f, v);
    // Normalise the movement vector and make it proportional to the speed per se
cond.
```

```
        movement = movement.normalized * speed * Time.deltaTime;
        // Move the player to it's current position plus the movement.
        playerRigidbody.MovePosition (transform.position + movement);
    }
    void Turning ()
    {
        // Create a ray from the mouse cursor on screen in the direction of the camera.
        Ray camRay = Camera.main.ScreenPointToRay (Input.mousePosition);
        // Create a RaycastHit variable to store information about what was hit by the ray.
        RaycastHit floorHit;
        // Perform the raycast and if it hits something on the floor layer...
        if (Physics.Raycast (camRay, out floorHit, camRayLength, floorMask)) {
            // Create a vector from the player to the point on the floor the raycast from the mouse hit.
            Vector3 playerToMouse = floorHit.point - transform.position;
            //Vector3 playerToMouse =transform.position;
            // Ensure the vector is entirely along the floor plane
            playerToMouse.y = 0f;
            // Create a quaternion (rotation) based on looking down the vector from the player to the mouse.
            Quaternion newRotatation = Quaternion.LookRotation (playerToMouse);
            // Set the player's rotation to this new rotation.
            playerRigidbody.MoveRotation (newRotatation);
        }
    }
    void Animating (float h, float v)
```

```
        {
            // Create a boolean that is true if either of the input axes is non-zero.
            bool running = h != 0f || v != 0f;
            // Tell the animator whether or not the player is walking.
            anim.SetBool ("IsRunning", running);
        }
    }
```

5.2.4 *PlayShoot*

```
using UnityEngine;
public class PlayerShoot : MonoBehaviour
{
  public int damagePerShot = 20; // The damage inflicted by each bullet.
  public float timeBetweenBullets = 0.15f;  // The time between each shot.
   public float timeBetweenGrenades = 1f; // The time between each grenade.
   public float range = 100f;      // The distance the gun can fire.
  float timer;  // A timer to determine when to fire.
   Ray shootRay = new Ray(); // A ray from the gun end forwards.
 RaycastHit shootHit;//A raycast hit to get information about what was hit.
  int shootableMask;// A layer mask so the raycast only hits things on the shootable
 layer.
   ParticleSystem gunParticles;
   LineRenderer gunLine;
   AudioSource gunAudio;
   Light gunLight;
   Animator anim;
  float effectsDisplayTime = 0.2f;  // The proportion of the timeBetweenBullets tha
t the effects will display for.
  public GameObject grenadePrefab;
```

```
    public GameObject bloodPrefab;
    public AudioClip shootClip;
    public AudioClip grenadePinPullClip;
    void Awake ()
    {
        shootableMask = LayerMask.GetMask ("Shootable");
        gunParticles = GetComponent<ParticleSystem> ();
        gunLine = GetComponent <LineRenderer> ();
        gunAudio = GetComponent<AudioSource> ();
        gunLight = GetComponent<Light> ();
        anim = GetComponentInParent <Animator>();
    }
    void Update (){
        anim.SetBool ("IsShooting", false);
        timer += Time.deltaTime;
       if(Input.GetMouseButton(0) && timer >= timeBetweenBullets && Time.time
Scale != 0)
        {
            gunAudio.clip = shootClip;
            Shoot ();
        }
       if(Input.GetMouseButtonDown(1) && timer >= timeBetweenGrenades && Ti
me.timeScale != 0 && GameManager.numberOfGrenades != 0)
        {
            gunAudio.clip = grenadePinPullClip;
            gunAudio.Play ();
        }
       if(Input.GetMouseButtonUp(1) && timer >= timeBetweenGrenades && Time
.timeScale != 0 && GameManager.numberOfGrenades != 0)
```

```
        {
            ThrowGrenade();
        }               // If the timer has exceeded the proportion of timeBetweenBullets t
hat the effects should be displayed for
        if(timer >= timeBetweenBullets * effectsDisplayTime)
        {
            DisableEffects ();
        }
    }
    public void DisableEffects ()
    {
        // Disable the line renderer and the light.
        gunLine.enabled = false;
        gunLight.enabled = false;
    }
    void Shoot ()
    {
        anim.SetBool ("IsShooting", true);
        // Reset the timer.
        timer = 0f;
        // Play the gun shot audioclip.
        gunAudio.Play ();
        // Enable the lights.
        gunLight.enabled = true
  // Stop the particles from playing if they were, then start the particles.
        gunParticles.Stop ();
        gunParticles.Play ();
        // Enable the line renderer and set it's first position to be the end of the gun.
        gunLine.enabled = true;
```

```
        gunLine.SetPosition (0, transform.position);
        // Set the shootRay so that it starts at the end of the gun and points forward fro
m the barrel.
        shootRay.origin = transform.position;
        shootRay.direction = transform.forward;
        // Perform the raycast against gameobjects on the shootable layer and if it hits
something...
        if(Physics.Raycast (shootRay, out shootHit, range, shootableMask))
        {
            // Try and find an EnemyHealth script on the gameobject hit.
           EnemyHealth enemyHealth = shootHit.collider.GetComponent <EnemyHeal
th> ();
            // If the EnemyHealth component exist...
            if(enemyHealth != null)
            {
                // ... the enemy should take damage.
                enemyHealth.TakeDamage (damagePerShot);
               Transform enemyTransform = enemyHealth.GetComponentInParent<Tra
nsform> ();
                Vector3 offset = new Vector3 (0, 1, 0);
                Vector3 enemyPosition = enemyTransform.position + offset;
               var blood = (GameObject)Instantiate (bloodPrefab, enemyPosition, enem
yTransform.rotation);
                Destroy (blood, 1f);
            }
            // Set the second position of the line renderer to the point the raycast hit.
            gunLine.SetPosition (1, shootHit.point);
        }
        // If the raycast didn't hit anything on the shootable layer...
```

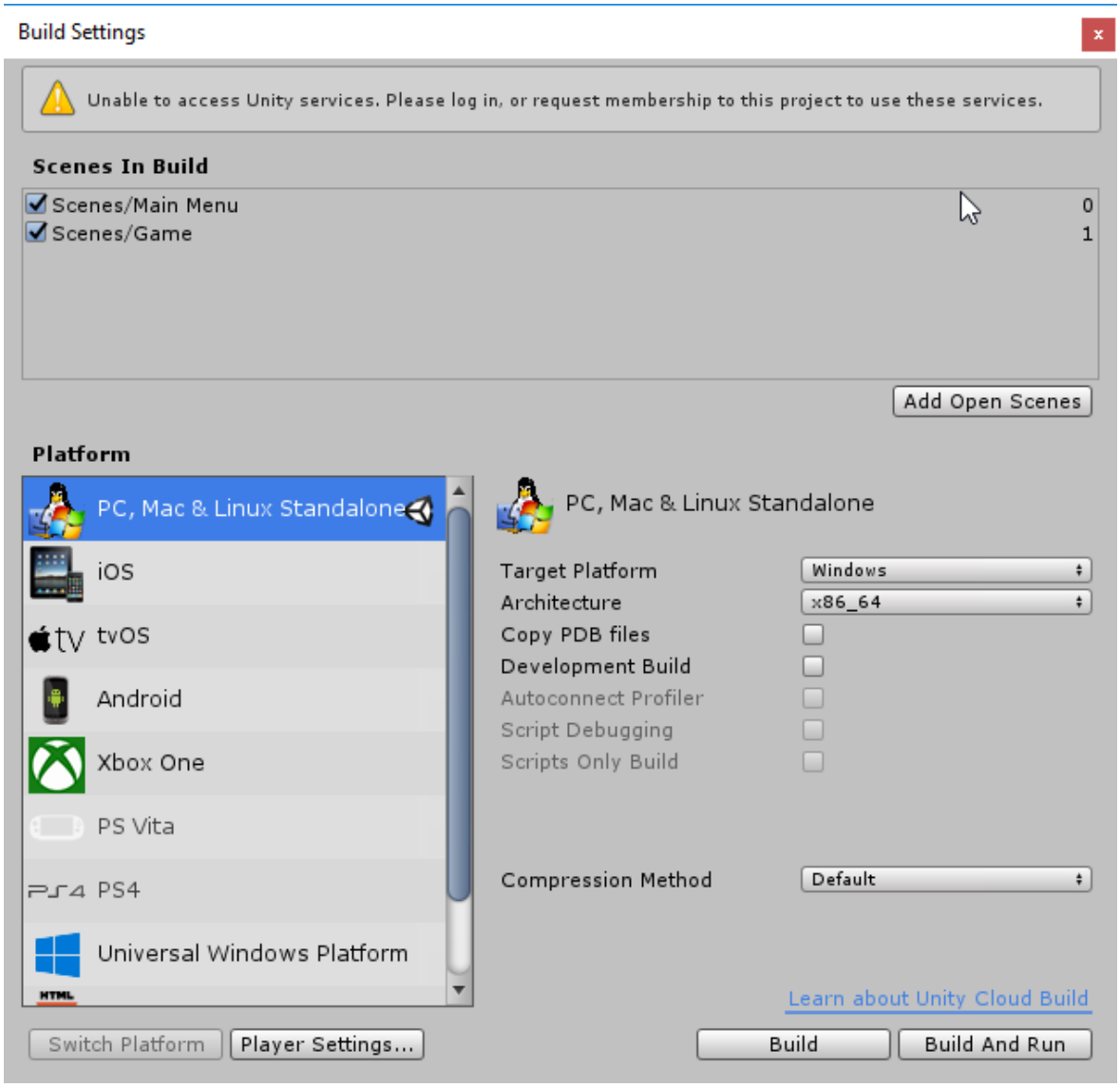
```
        else
        {
            // ... set the second position of the line renderer to the fullest extent of the gu
n's range.
            gunLine.SetPosition (1, shootRay.origin + shootRay.direction * range);
        }

    }
    void ThrowGrenade()
    {
        GameManager.numberOfGrenades  -= 1;
        timer = 0f;
       var grenade = (GameObject)Instantiate (grenadePrefab, gameObject.transform.
position, gameObject.transform.rotation);
       grenade.GetComponent<Rigidbody> ().velocity = grenade.transform.forward *
 20;    }
}
```

5.3 Build setting

Unity hỗ trỡ Biuld đa nền tảng trên các môi trường như Window,IOS…..

Ta sẽ chọn vào mục player setting để chỉnh sửa tên và hình ảnh cho game của mình .



Hình 36. Build Settings.

Chương 6 . KẾT LUẬN

Sau nhiều ngày suy nghĩ kĩ để chọn đề tài cho đồ án tốt nghiệp, cuối cùng em cũng đã chọn được đề tài làm game 3D trên Unity Game Run Now.

Bước đầu tìm hiểu về Unity bằng cách làm mấy game thông qua hướng dẫn trên Youtube để hiểu rõ hơn về nó.

Một số game em làm để tham khảm khảo là : Flappy bird,Ufo,Rool a roll,..

Trong thời gian làm đề tài em cũng có một số những vấn đề đạt được và chưa đạt được.

Đạt Được

Hiểu được cấu trúc, cách thức hoạt động, cách thức phát triển ứng dụng trên Unity

Tổ chức load các màn tương ứng thông qua các button

Xây dựng các chức năng như điểm, máu, màn chơi.. thông qua ngôn ngữ lập trình C# trên MonoDevelop

Tạo âm thanh cho cho trò chơi và 1 số chức năng khác.

Ngoài những thứ đạt được thì em còn rất nhiều những vấn đề chưa đạt được.

Chưa Đạt Được

Do kiến thức của em về Unity còn hạn hẹp và quá trình thực tập full time đã hạn chế thời gian của em nên các chức năng trong game còn nghèo nàn.

Chưa xây dựng được hoàn chỉnh 1 game như mong muốn.

Các Script trong Unity tổ chức chưa thực sự rõ ràng.

Thuận Lợi

Vận dụng được các kiến thức mà em đã học trong tường để làm đồ án này.

Được sự chỉ dạy tận tình của cô và các bạn.

Đồ họa Unity có rất nhiều assets free cho mọi người.

Tài liệu rất nhiều trên Internet.

Khó Khăn

Unit là một môi trường lập trình mới đối với em nên khi bắt tay vào làm thì em không biết làm từ đâu và cảm thấy bỡ ngỡ

Do quá trình thực tập chiếm hết phần thời trong tuần.

Kinh Nghiệm Rút Ra

Phải biết sắp xếp thời gian 1 cách hợp lý nếu không sẽ gặp rất nhiều khó khăn.

Để hoàn thành tốt 1 game không thể làm 1 mình mà phải biết chia sẻ và phân công công việc cho mọi người.

Làm việc phải tập trung mới có thể hoàn thành tốt được.

Hướng Phát Triển

Tối ưu hóa để game có thể chạy nhanh hơn.

Tăng thêm màn chơi để người chơi có nhiều lựa chọn.

Thêm vũ khí, dạn dược, nâng cấp vũ khí.

Cung cấp thêm công cụ để người chơi có thể chọn và chỉnh sửa tạo riêng cho mình những nhân vật, hiệu ứng âm thanh….

Chương 7 . TÀI LIỆU THAM KHẢO

❖ Tài liệu trích dẫn từ Internet:

- Hướng dẫn làm một số game mini tương ứng với hiệu ứng của Unity3d

  https://unity3d.com/learn/tutorials

- Local docs của Unity hướng dẫn và giải thích một số function

  file:///F:/Wanpi-su/Unity/Editor/Data/Documentation/en/Manual/ScriptingSection.html

  OR Link online of unity

  https://docs.unity3d.com/Manual/index.html

- Một số trang trang hướng dẫn viết một game mini nhỏ như của Trung Tâm Đào Tạo Tin Học Khoa Phạm

  https://www.youtube.com/watch?v=_qPRDKmpEKk&list=PLzrVYRai0riRwq876NCjZuulv5BjuDCBk OR Ploxer Games

  https://www.youtube.com/watch?v=mMlY4MXSrv8&list=PLqO_dK1hz3TMXDRmW-aNHGosp5LzNRhW5

- Hướng dẫn sử dụng asset

  https://learn.unity.com/tutorial/asset-store-basics#5c7f8528edbc2a002053b445

- Hướng dẫn tạo một FPS

  https://unity.com/fps-sample

  https://github.com/jiankaiwang/FirstPersonController

  https://www.raywenderlich.com/738-introduction-to-ufps-unity-fps-tutorial

- Hướng dẫn viết AI

  https://answers.unity.com/questions/282228/zombie-ai-tutorial-1.html

  https://www.youtube.com/watch?v=nlCadtsXVY8