|  |  |
| --- | --- |
| NGUYỄN HỮU TĂNG CÔNG NGHỆ THÔNG TIN | **TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI**  **KHOA CÔNG NGHỆ THÔNG TIN**  **======\*\*\*======**  **E:\dh-cn-hanoi.png**  **ĐỒ ÁN, KHÓA LUẬN TỐT NGHIỆP ĐẠI HỌC**  **CÔNG NGHỆ THÔNG TIN**  **Đề tài: Xây dựng ứng dụng game 2D Sword Hunter bằng Unity**  **CBHD :** ThS. Nguyễn Đức Lưu  **Sinh viên:** Nguyễn Hữu Tăng  **Mã SV :** 2019606645  **Lớp :** CNTT06 **Khóa:** K14  **Hà Nội – Năm 2023** |

LỜI CẢM ƠN

Được sự phân công của Khoa Công nghệ thông tin Trường Đại học Công nghiệp Hà Nội, và dưới sự hướng dẫn của Thầy giáo hướng dẫn Ths. Nguyễn Đức Lưu, em đã hoàn thành được đề tài: “Xây dựng ứng dụng game 2D Sword Hunter bằng Unity”.

Để hoàn thành được kháo luận này, em xin chân thành cảm ơn Trường Đại học Công nghiệp Hà Nội, cùng với các thầy, cô khoa Công nghệ thông tin đã day dỗ em trong quá trình học tập và rèn luyện; nhờ công ơn của Nhà Trường và thầy cô đã giúp em có được những kiến thức không chỉ để hoàn thành khóa luận này mà còn có thể áp dụng vào trong công việc. Đặc biệt, em xin gửi lời cảm ơn tới thầy **ThS. Nguyễn Đức Lưu** đã tận tình hướng dẫn, chỉ bảo và giúp đỡ em trong suốt quá trình thực hiện khóa luận này.

Em đã cố gắng rất nhiều để thực hiện đề tài một cách tốt nhất. Em rất mong nhận được sự góp ý của các thầy cô để hoàn thiện đồ án này tốt hơn.

*Em xin chân thành cảm ơn!*

Sinh viên thực hiện

Nguyễn Hữu Tăng

MỤC LỤC

[LỜI CẢM ƠN 2](#_Toc134233854)

[MỤC LỤC 3](#_Toc134233855)

[DANH MỤC HÌNH ẢNH 4](#_Toc134233856)

[MỞ ĐẦU 7](#_Toc134233857)

[CHƯƠNG 1: LẬP TRÌNH C# TRONG UNITY 9](#_Toc134233858)

[1.1. Kịch bản (Scripting) trong Unity 9](#_Toc134233859)

[1.2. Ngôn ngữ được sử dung trong Unity 10](#_Toc134233860)

[1.3. Biến, Hàm, Lớp và cách sử dụng 11](#_Toc134233861)

[1.4. GameObject – Thao tác với C# Script 19](#_Toc134233862)

[CHƯƠNG 2: LẬP TRÌNH GAME UNITY 26](#_Toc134233863)

[2.1. Tổng quan về Unity 26](#_Toc134233864)

[2.2. Các thành phần cơ bản trong Unity 28](#_Toc134233865)

[2.3. Một số bước lập trình game 2D với Unity 34](#_Toc134233866)

[CHƯƠNG 3: ỨNG DỤNG GAME 2D SWORD HUNTER 38](#_Toc134233867)

[3.1. Một số Scene trong Game 38](#_Toc134233868)

[3.2. Nhân vật Player 44](#_Toc134233869)

[3.3. Một số thành phần trong game 54](#_Toc134233870)

[KẾT LUẬN VÀ KIẾN NGHỊ 57](#_Toc134233871)

[TÀI LIỆU THAM KHẢO 58](#_Toc134233872)

DANH MỤC HÌNH ẢNH

[Hình 1: Một số mã code trong Unity 10](#_Toc134233808)

[Hình 2: Ví dụ về biến (Variables) 12](#_Toc134233809)

[Hình 3: Ví dụ các loại của biến 13](#_Toc134233810)

[Hình 4: Hiển thị các biến public trong Inspector 14](#_Toc134233811)

[Hình 5: Một số hàm cơ bản 15](#_Toc134233812)

[Hình 6: Ví dụ về hàm 16](#_Toc134233813)

[Hình 7: Ví dụ về lớp 17](#_Toc134233814)

[Hình 8: Ví dụ về lớp 18](#_Toc134233815)

[Hình 9: Tạo Project 28](#_Toc134233816)

[Hình 10: Tạo project 28](#_Toc134233817)

[Hình 11: Định dạng file 30](#_Toc134233818)

[Hình 12: Tạo Folder 31](#_Toc134233819)

[Hình 13: Tạo Scene 32](#_Toc134233820)

[Hình 14: Viết kịch bản game 35](#_Toc134233821)

[Hình 15: Ví dụ về đồ họa 2D trong Unity 35](#_Toc134233822)

[Hình 16: Màn hình Logo 38](#_Toc134233823)

[Hình 17: Màn hình Loading 38](#_Toc134233824)

[Hình 18: Màn hình Loading 39](#_Toc134233825)

[Hình 19: Màn hình Menu 39](#_Toc134233826)

[Hình 20: Một số trang bị trong game 40](#_Toc134233827)

[Hình 21: Shop trang bị 40](#_Toc134233828)

[Hình 22: Shop trang bị 41](#_Toc134233829)

[Hình 23: Màn hình Map 1 41](#_Toc134233830)

[Hình 24: Màn hình Map 2 42](#_Toc134233831)

[Hình 25: Màn hình Map 3 42](#_Toc134233832)

[Hình 26: Màn hình Pause 43](#_Toc134233833)

[Hình 27:Màn hình Game Over 43](#_Toc134233834)

[Hình 28: Nhân vật Player 44](#_Toc134233835)

[Hình 29: Player 44](#_Toc134233836)

[Hình 30: Component Player 45](#_Toc134233837)

[Hình 31: Skiill và Coins 46](#_Toc134233838)

[Hình 32: Skill 1: Đánh thường 47](#_Toc134233839)

[Hình 33: Skill 2: Cầu lửa 47](#_Toc134233840)

[Hình 34: Skill 3: Lửa bùng nổ 47](#_Toc134233841)

[Hình 35: Moving Player 48](#_Toc134233842)

[Hình 36: Change Health 49](#_Toc134233843)

[Hình 37: Animation 50](#_Toc134233844)

[Hình 38: Parameters 51](#_Toc134233845)

[Hình 39: Tạo Blend Tree 51](#_Toc134233846)

[Hình 40: Blend Tree 52](#_Toc134233847)

[Hình 41: Script Animation 53](#_Toc134233848)

[Hình 42: Một số NPC 54](#_Toc134233849)

[Hình 43: Sound 54](#_Toc134233850)

[Hình 44: Quái và Boss 54](#_Toc134233851)

[Hình 45: Enemy Moving 55](#_Toc134233852)

[Hình 46: Enemy Skill 56](#_Toc134233853)

MỞ ĐẦU

Báo cáo của Newzoo tính đến tháng 11.2022 cho thấy quy mô ngành game toàn cầu đạt 184,4 tỉ USD, với doanh thu ngành game di động ước tính đạt 92,2 tỉ USD và chiếm khoảng 50% thị phần. Trong số này, một số khu vực nổi bật có tốc độ tăng trưởng thị trường game di động nhanh trong năm 2022 gồm Mỹ La tinh, Trung Đông, Đài Loan & Hồng Kong, Đông Nam Á. Doanh thu game di động tại thị trường Đông Nam Á đạt 4,5 tỉ USD, chỉ sau Mỹ La tinh.

Mặc dù xếp thứ ba về doanh thu trong năm 2022 khi đạt 507 triệu USD từ 54,6 triệu người chơi, nhưng quy mô doanh thu game di động ở thị trường Việt Nam vẫn còn duy trì khoảng cách khá xa so với Thái Lan - vốn đạt doanh thu 763 triệu USD với 38,3 triệu người chơi.

Nếu nhìn vào số dân và lượng tải game ở Việt Nam có thể thấy trung bình số lượng người chơi game ở Việt Nam là khá lớn. Việt Nam có nội lực và tiềm năng cần khai phá trong lĩnh vực game.

Thị trường game Việt Nam vẫn còn nhỏ bé và chưa được nhiều tập đoàn lớn quan tâm. Tuy nhiên, Việt Nam đang trong thời điểm chín muồi để khai thác tiềm năng phát triển cả về khách hàng và lực lượng công nghệ game phát triển mạnh mẽ, trong đó có lĩnh vực thể thao điện tử. Thể thao điện tử đang được cộng đồng đánh rất cao. Game không chỉ là trò chơi giải trí thông thường mà còn có tính thể thao.

Chính vì lý do đó mà em quyết định **“Xây dựng ứng dụng game 2D Sword Hunter bằng Unity”**

**Mục đích đề tài**

Đề tài **“Xây dựng ứng dụng game 2D Sword Hunter bằng Unity”** nhằm xây dựng một ứng dụng game 2D dựa trên công cụ có sẵn là Unity.

**Bố cục đề tài**

Nội dung chính được chia làm 3 chương:

* **Chương 1: Lập trình C# trong Unity**

Kiến thức cơ bản về C# trong lập trình game Unity: như biến, hàm và lớp, cũng như cách sử dụng chúng.

* **Chương 2: Lập trình game Unity**

Kiến thức cơ bản về Unity và cách làm việc với Unity.

* **Chương 3: Ứng dụng game 2D Sword Hunter**

Các thành phần cơ bản trong ứng dụng game 2D Sword Hunter.

1. LẬP TRÌNH C# TRONG UNITY
   1. Kịch bản (Scripting) trong Unity
      1. Scripting là gì?

Scripting (viết kịch bản) giúp cho các GameObjects biết cách hành xử; các tập lệnh và thành phần đính kèm với GameObjects, và cách chúng tương tác với nhau sẽ tạo nên lối chơi cho game. Bây giờ, viết kịch bản trong Unity khác với lập trình thuần túy. Nếu thực hiện một số chương trình thuần túy, ví dụ: Tạo một ứng dụng đang chạy, có thể nhận ra rằng trong Unity không cần tạo mã chạy ứng dụng, vì Unity làm việc đó giúp cho người lập trình. Thay vào đó, nên tập trung vào lối chơi trong kịch bản của game.

Unity chạy trong một vòng lặp lớn. Nó đọc tất cả dữ liệu trong một cảnh trò chơi (game scene). Ví dụ: nó đọc qua ánh sáng, mắt lưới, hành vi là gì và nó xử lý tất cả thông tin này thay cho chúng ta.

Ví dụ, nếu truyền hình như ở Bắc Mỹ, sẽ có 29,5 frame/sec, Unity cần phải làm được điều tương tự. Nó chạy từng khung hình rời rạc, nối tiếp nhau. Lập trình viên chỉ đạo Unity bằng các hướng dẫn mà họ viết trong tập lệnh của họ và Unity thực thi chúng hết khung này đến khung khác nhanh nhất có thể.

Đạt được tốc độ khung hình cao có nghĩa là không chỉ trò chơi trông trôi chảy hơn mà các tập lệnh cũng sẽ được thực thi thường xuyên hơn, giúp điều khiển phản ứng nhanh hơn.

* 1. Ngôn ngữ được sử dung trong Unity

Một Tập lệnh (Script) phải được đính kèm vào một GameObject trong cảnh (scene) để được gọi ra bởi Unity. Các tập lệnh được viết bằng một ngôn ngữ đặc biệt mà Unity có thể hiểu được. Và, thông qua ngôn ngữ này, chúng ta có thể nói chuyện với máy tính và đưa ra hướng dẫn cho nó hiểu.

Ngôn ngữ được sử dụng trong Unity được gọi là C# (phát âm là C-sharp). Tất cả các ngôn ngữ mà Unity hoạt động đều là ngôn ngữ kịch bản hướng đối tượng. Giống như bất kỳ ngôn ngữ nào, ngôn ngữ kịch bản có cú pháp hoặc các phần của ngôn ngữ và các phần chính được gọi là biến, hàm và lớp (variables, functions, and classes).

Các phiên bản Unity cho đến trước 2017.3, sẽ có một trình soạn thảo văn bản có tên là MonoDevelop: nó có thể giúp chúng ta hoàn thành mã của mình, nó sẽ cho chúng ta biết nếu chúng ta viết sai một đoạn mã tại ví trị cụ thể, và cho phép chúng ta đi đường tắt đến vị trí sai đó. Kể từ phiên bản 2018.1, cũng có thể sử dụng trình soạn thảo mã như Visual Studio cho Unity Community hoặc các trình soạn thảo văn bản khác như Visual Studio, Notepad hoặc Sublime text.



Hình : Một số mã code trong Unity

Có thể thấy, có các biến, hàm và lớp.

* 1. Biến, Hàm, Lớp và cách sử dụng
     1. Gồm những gì?

**Các biến (Variables)** giữ các giá trị và tham chiếu đến các đối tượng - object (có thể xem các đối tượng là các biến “lớn hơn”). Chúng giống như một cái hộp chứa đựng thứ gì đó để chúng ta sử dụng. Các biến bắt đầu bằng một chữ cái viết thường.

**Hàm (Functions)** là tập hợp các mã được đối chiếu và thao tác các biến này. Các hàm bắt đầu bằng một chữ cái viết hoa. Chúng ta tổ chức mã trong các hàm để chúng có thể dễ dàng gọi đến để sử dụng lại nhiều lần trong các phần khác nhau của chương trình.

**Các lớp (Classes)** là một cách để cấu trúc mã dùng để bọc tập hợp các biến và hàm lại với nhau để tạo một mẫu xác định các thuộc tính của một đối tượng.

Scripting chủ yếu so sánh các đối tượng này với trạng thái và giá trị hiện tại của chúng. Nó dựa trên logic xác định kết quả hoặc giải pháp.

* + 1. Variables

Trong Unity, các tập lệnh bắt đầu bằng cách đặt các công cụ cần ở trên cùng và điều này thường là bằng cách khai báo các biến. Có thể thấy các biến được khai báo ở đây với từ khóa hiển thị “public” (công khai) hoặc "private" (riêng tư) ở phía trước, theo sau là loại (type) và tên (name).



Hình : Ví dụ về biến (Variables)

Khi khai báo các biến, có một số loại khả năng hiển thị, nhưng hai loại quan trọng nhất là public và private.

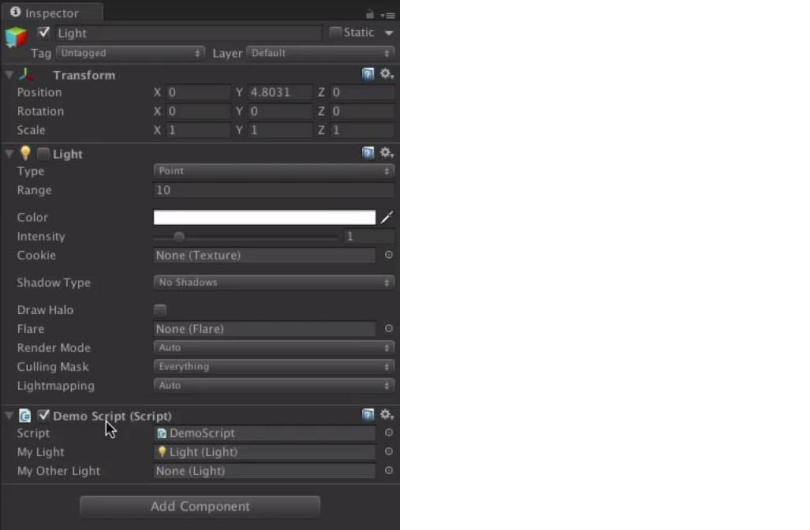
Nếu tạo tập lệnh với văn bản trên trong trình chỉnh sửa mã, sau đó quay lại Unity và gán tập lệnh cho GameObject, có thể truy cập và xem biến được khai báo là công khai trong Trình kiểm tra (Inspector), nhưng không thể nhìn thấy cái private. Và đó là bởi vì những gì được định nghĩa là “private” chỉ có thể được truy cập trong tập lệnh cụ thể này, trong lớp cụ thể này.

Nếu công khai điều này, thì các tập lệnh khác và các lớp khác có thể truy cập được và có thể thay đổi trong Trình kiểm tra từ trình chỉnh sửa Unity. Vì vậy, điều đó có nghĩa là những người khác có thể truy cập và thay đổi giá trị của nó.

Có nhiều lý do để lựa chọn giữa private hoặc public. Các biến riêng tư cho phép mã an toàn hơn, vì biết rằng giá trị của các biến đó chỉ có thể được thay đổi bên trong lớp đó. Điều này làm cho việc gỡ lỗi và bảo trì mã dễ dàng hơn.

Nếu lựa chọn “public” và gặp sự cố, chúng ta cần xem xét bên trong toàn bộ cơ sở mã để theo dõi nguồn vì bất kỳ đối tượng nào khác đều có quyền truy cập vào biến đó. Tuy nhiên, nếu muốn các đối tượng giao tiếp với nhau, cần công khai một số biến (hoặc hàm).

Một khía cạnh quan trọng khác của các biến là loại (type). Loại xác định loại giá trị mà biến đang lưu giữ trong bộ nhớ, ví dụ: nó có thể là một số (number), văn bản (text) hoặc các loại phức tạp hơn, giống như các loại trong hình bên dưới: Transform, Light và Demo Script trong hình bên dưới thực tế là các tham chiếu đến Thành phần (Components). Unity cần biết loại đối tượng đó là gì để biết cách xử lý.



Hình : Ví dụ các loại của biến

Một điều quan trọng khác về các biến là tên (name). Điều chính mà chúng ta cần nhớ về cách đặt tên biến là nó không được bắt đầu bằng một số và không được chứa khoảng trắng. Do đó, có một số phong cách viết tên. Trong C#, quy ước đặt tên là camelCase: tên bắt đầu bằng một chữ cái viết thường và thêm các từ, không có dấu cách, tiếp theo bắt đầu bằng một chữ cái in hoa, ví dụ: "myLight".

Khi Unity biên dịch tập lệnh, nó sẽ hiển thị các biến công khai trong trình chỉnh sửa.



Hình : Hiển thị các biến public trong Inspector

* + 1. Functions

Các tập lệnh thao tác với các biến bằng cách sử dụng các hàm. Có một số chức năng của hàm chạy tự động bên trong Unity.



Hình : Một số hàm cơ bản

**Awake** chỉ được gọi một lần khi GameObject với thành phần đó được khởi tạo. Nếu một GameObject không hoạt động, thì nó sẽ không được gọi cho đến khi nó được kích hoạt. Tuy nhiên, Awake được gọi ngay cả khi GameObject đang hoạt động nhưng thành phần này không được bật (với hộp kiểm nhỏ bên cạnh tên của nó). Có thể sử dụng Awake để khởi tạo tất cả các biến mà cần gán giá trị.

**Start** – giống như Awake, Start sẽ được gọi nếu GameObject đang hoạt động, nhưng chỉ khi thành phần này được bật.

**Update** được gọi một lần cho mỗi khung hình. Đây là nơi đặt mã để xác định logic chạy liên tục, chẳng hạn như hoạt ảnh, trí tuệ nhân tạo và các phần khác của trò chơi phải được cập nhật liên tục.

**FixedUpdate** là khi muốn thực hiện công việc vật lý.

**LateUpdate** là một chức năng tương tự như Update, nhưng LateUpdate được gọi ở cuối khung hình. Unity sẽ xem xét tất cả các đối tượng trò chơi, tìm tất cả các Updates và gọi các LateUpdates. Điều này tốt cho những thứ như máy ảnh. Giả sử muốn di chuyển một nhân vật trong trò chơi. Và rồi anh ấy đụng phải một nhân vật khác và kết thúc ở một vị trí khác. Nếu chúng ta di chuyển máy ảnh cùng lúc với nhân vật, sẽ có tiếng lắc lư và máy ảnh sẽ không ở đúng vị trí cần thiết. Vì vậy, về cơ bản, đó là một vòng lặp thứ hai rất tiện dụng.

**Viết hàm (Writing functions)**

Khi viết một hàm, hãy nhớ rằng các hàm bắt đầu với kiểu trả về của hàm ở đầu, tiếp theo là tên của hàm và sau đó là các tham số trong ngoặc đơn (nếu có). Tên hàm bắt đầu bằng chữ in hoa và phần thân của hàm nằm giữa dấu ngoặc nhọn. Đây là một ví dụ về cách viết một hàm:

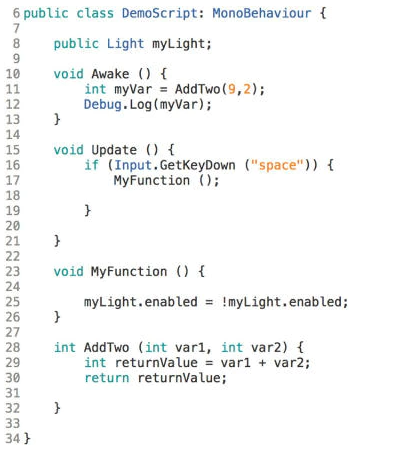


Hình : Ví dụ về hàm

Các hàm có thể thực hiện các phép tính và sau đó trả về một giá trị. Có thể yêu cầu một hàm làm điều gì đó, xử lý thông tin, sau đó trả về (return). Nếu sử dụng loại "void", thì sẽ không trả về bất kỳ thứ gì.

* + 1. Classes

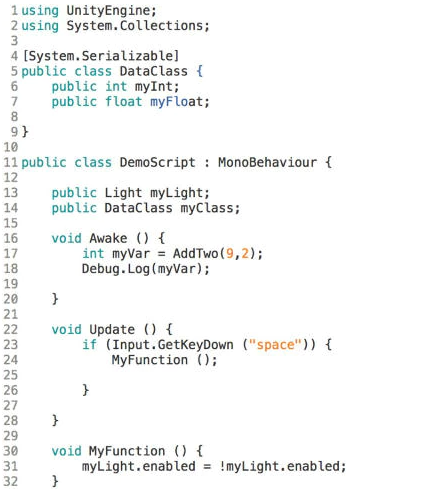
Các lớp là tập hợp của các biến và hàm. Ví dụ: tập lệnh này là một lớp:



Hình : Ví dụ về lớp

Hãy nhớ rằng tên lớp phải khớp với tên tệp của tập lệnh C# để nó hoạt động. Và sau đó để được gắn vào một GameObject, nó phải xuất phát từ một lớp khác có tên là MonoBehaviour, lớp này sẽ tự động được đặt ở đó khi tạo tập lệnh lần đầu tiên. Các lớp cũng có thể là công khai - public hoặc riêng tư - private.

Trong Unity, nếu tạo một lớp tùy chỉnh, như trong ví dụ bên dưới, phải yêu cầu nó tuần tự hóa nó. Điều này có nghĩa là nó sẽ được chuyển đổi thành dữ liệu đơn giản mà Unity có thể xem trong trình kiểm tra. Khi làm điều đó, nó sẽ thấy rằng có lớp sẽ xuất hiện trong trình kiểm tra.



Hình : Ví dụ về lớp

Các biến, hàm và lớp chỉ là những kiến ​​thức cơ bản khi bắt đầu viết mã trong Unity.

* 1. GameObject – Thao tác với C# Script

GameObject là khái niệm cơ bản trong Unity, mọi đối tượng trong game, từ Camera đến các đối tượng UI hay các đối tượng khác trong game được hiển thị trong cửa sổ Hierarchy đều là các GameObject.

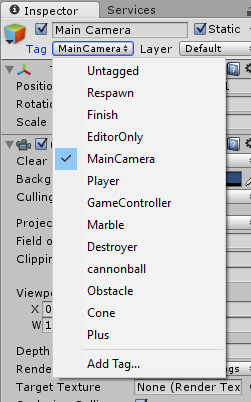
Các loại đối tượng khác nhau sẽ có những thao tác đặc thù riêng, nhưng là GameObject đều có những phương thức và thuộc tính chung.

* + 1. Thuộc tính và hàm cơ bản

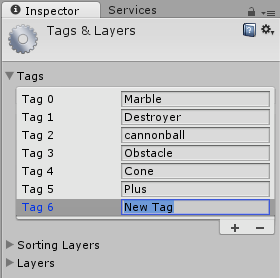
**Tag – Name**

Unity sử dụng 2 cơ chế để lưu trữ và quản lý danh sách các GameObject, được hiển thị trong cửa sổ Hierarchy.

* Thuộc tính name được kế thừa lại từ lớp Object, lưu trữ tên của đối tượng được hiển thị ở Hierarchy.
* Thuộc tính tag được sử dụng để phân loại và định danh các GameObject, phải được định nghĩa sẵn trước khi sử dụng. Các đối tượng chưa được phân loại sẽ được gán mặc định tag “Untagged”. Để định nghĩa tag, sử dụng Tag Manager tại cửa sổ Inspector của bất kỳ đối tượng nào.



Chọn nút Add Tag, bấm chọn dấu + để thêm tag mới và đặt tên cho nó.



Name và Tag được sử dụng nhiều trong Script để chọn được đối tượng cần thao tác.

\* Name không bắt buộc khác nhau giữa các đối tượng nhưng nên đặt tên cho mỗi đối tượng khác nhau để dễ phân biệt và xử lý trong Script. Cách đặt tên phổ biến là 1 tên chung và 1 index cho từng loại đối tượng, ví dụ “Monster3”.

**Layer**

Tại Tag Manager, có thể tạo ra các Layer và Sorting Layer để thuận tiện trong việc quản lý các đối tượng.

* Sorting Layer được sử dụng để phân lớp hiển thị các đối tượng và sử dụng với thành phần SpriteRenderer.
* Layer được sử dụng để phân lớp đối tượng và tùy chọn hiển thị trong Camera, một số lớp được Unity tạo sẵn cơ bản là đủ với các nhu cầu thông thường của lập trình viên.

**Active**

2 thuộc tính activeInhierarchy và activeSelf được sử dụng để kiểm tra trạng thái hoạt động của đối tượng. Đối với việc thiết lập trạng thái, sử dụng hàm SetActive với giá trị truyền vào là true hay false, là trạng thái cần thiết để thiết lập cho đối tượng đó. Nếu đối tượng cha có trạng thái active là false thì dù có thay đổi trạng thái của đối tượng con chúng vẫn trả về giá trị false.

**IsStatic**

1 số đối tượng trong game như bản đồ hay các chướng ngại vật tĩnh đều không có khả năng di chuyển, do đó nên thiết lập trạng thái Static cho các đối tượng đó để tiết kiệm tài nguyên.

Các đối tượng Static sẽ được tiền xử lý và gom chung thành 1 đối tượng gọi là Batching object, nhờ đó tiết kiệm được 1 số lượng Draw call (Draw call là số lần render 1 đối tượng, mỗi lần render là 1 draw call).

Tương tự như trạng thái Active của đối tượng, trong Unity Editor có 1 checkbox dùng thiết lập trạng thái Static của đối tượng, thuộc tính isStatic dùng để kiểm tra trạng thái tĩnh của đối tượng, từ đó bỏ qua các thao tác xử lý với đối tượng này.

* + 1. Thao tác với Component

**Transform**

Transform là 1 thành phần bắt buộc, không thể thêm hoặc xoá đối với bất kì GameObject nào.

Transform lưu trữ các thông tin về vị trí, phép quay và tỷ lệ của đối tượng. Ngoài ra Transform còn có 1 số thuộc tính và phương thức hay để thao tác với GameObject.

**GetComponent và GetComponents**

* GetComponent trả về component đầu tiên cần tìm của đối tượng.
* GetComponents trả về 1 danh sách toàn bộ các component cần tìm, nếu không tìm được sẽ trả về null hoặc mảng không có phần tử tương ứng.

Nguyên mẫu của hàm như sau:

public Component GetComponent(Type type);

public Component GetComponent(string type);

Hàm có thể nhận vào giá trị kiểu Type hoặc kiểu string đều được chấp nhận. Để tối ưu nên sử dụng kiểu Type trong hầu hết mọi trường hợp.

Cú pháp khác để lấy 1 thành phần với Type như sau:

public Type GetComponent<Type>();

Cú pháp này sử dụng template để lấy ra các component, đây là cách thông dụng trong C# Script.

**Tìm kiếm Component trong các đối tượng cha hoặc con**

Unity cũng cung cấp sẵn 1 số phương thức giúp thao tác được với component thuộc đối tượng cha hoặc con của nó. Cách thức hoạt động tương tự như GetComponent và GetComponents.

Các hàm như sau:

* GetComponentInChildren
* GetComponentsInChildren
* GetComponentInParent
* GetComponentsInParent

Các hàm này chỉ có thể được sử dụng với Type của component, kiểu string không được chấp nhận.

* + 1. Tìm kiếm GameObject

**Tìm kiếm với name**

Hàm Find được Unity thiết kế để tìm kiếm 1 đối tượng trong cửa sổ Hierarchy, cú pháp của hàm như sau:

public static GameObject Find(string name);

* Hàm sẽ trả về giá trị GameObject là chính đối tượng được tìm thấy.
* Nếu không có đối tượng có tên cần tìm, hàm sẽ trả về null.

Chuỗi name truyền vào chính là tên của đối tượng cần tìm, nếu chuỗi ký tự có chứa ký hiệu /, Unity sẽ xem đó là đường dẫn của đối tượng trong cửa sổ Hierarchy.

*\* Hàm chỉ trả về đối tượng được active trong cửa sổ Hierarchy, mọi đối tượng khác cho dù có tên đúng yêu cầu nhưng chưa được active cũng bị bỏ qua.*

**Tìm kiếm với tag**

Sử dụng hàm FindGameObjectWithTag hoặc FindGameObjectsWithTag tương ứng để tìm 1 hay toàn bộ các đối tượng có tag cần tìm, cách sử dụng tương tự như hàm Find đề cập ở trên, sử dụng tag để tìm kiếm đối tượng.

GameObject stdioObject = GameObject.FindGameObjectWithTag("StdioObject");

*\* Tìm kiếm với name và tag đều có thể cho ra kết quả giống nhau, nhưng về chi phí cần sử dụng thì tìm kiếm với tag sẽ tối ưu hơn. Do đó nên sử dụng tag cho việc tìm kiếm nếu khả thi.*

* + 1. Khởi tạo và giải phóng GameObject
       1. Khởi tạo

**CreatePrimitive**

Khởi tạo 1 đối tượng được định nghĩa sẵn bởi Unity, đối tượng được tạo ra sẽ có sẵn Mesh Renderer và Collider tương ứng với hình dạng của đối tượng, cách sử dụng hàm như sau:

GameObject plane = GameObject.CreatePrimitive(PrimitiveType.Plane);

GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);

cube.transform.position = new Vector3(0, 0.5f, 0);

GameObject sphere = GameObject.CreatePrimitive(PrimitiveType.Sphere);

sphere.transform.position = new Vector3(0, 1.5f, 0);

Danh sách các PrimitiveType được hỗ trợ:

|  |  |
| --- | --- |
| Sphere | Đối tượng hình cầu, bán kính bằng 1. |
| Capsule | Đối tượng hình capsule. |
| Cylinder | Đối tượng hình trụ. |
| Cube | Tạo ra 1 khối lập phương có độ dài cạnh bằng 1. |
| Plane | Tạo ra 1 mặt phẳng, được sử dụng trong việc xây dựng map. |
| Quad | Đối tượng quad (4 điểm trong không gian). |

**Instantiate**

Instantiate được sử dụng để clone 1 đối tượng được lập trình viên xây dựng sẵn. Nguyên mẫu hàm như sau:

public static Object Instantiate(Object original, Vector3 position, Quaternion rotation);

public static Object Instantiate(Object original);

Nếu không truyền vào position và rotation, hàm sẽ lấy các giá trị của đối tượng cơ sở để gán cho đối tượng mới, hoặc trả về Vector3.zero và Quaternion.identity nếu không khả dụng.

Khi sử dụng Instantiate, toàn bộ các đối tượng con và cấu trúc sẽ được sao chép với đúng tính chất của chúng. Tuy nhiên, đối tượng cha (nếu có) sẽ được đặt là null. Do đó đối tượng được clone có thể sẽ không nằm chung vị trí với đối tượng ban đầu.

Instantiate cũng sao chép được các đối tượng inactive. Tính chất này sẽ được giữ lại và bản sao được tạo ra cũng sẽ có trạng thái inactive.

* + - 1. Giải phóng

**Destroy**

Hàm Destroy được sử dụng để giải phóng 1 GameObject, component hay tài nguyên được sử dụng trong game. Hàm nhận vào giá trị là đối tượng cần giải phóng, cùng với thời gian delay (mặc định là 0). Việc huỷ đối tượng sẽ được delay cho đến khi kết thúc hàm Update, nhưng đối tượng sẽ được huỷ trước quá trình render.

Dưới đây là ví dụ giúp hiểu về chức năng và cách sử dụng hàm Destroy:

// Phá hủy đối tượng trò chơi hiện tại

Destroy (gameObject);

// Xóa phiên bản tập lệnh này khỏi đối tượng trò chơi

Destroy (this);

// Xóa phần cứng khỏi đối tượng trò chơi

Destroy (GetComponent<BoxCollider>());

// Phá hủy đối tượng trò chơi trong 5 giây sau khi tải đối tượng

Destroy (gameObject, 5);

**DontDestroyOnLoad**

Khi thực hiện chuyển scene, các đối tượng ở scene cũ sẽ được giải phóng hoàn toàn, hàm này giúp chỉ định 1 đối tượng sẽ không bị giải phóng khi chuyển scene, cách sử dụng hàm như sau:

DontDestroyOnLoad(targetObject);

targetObject chính là đối tượng sẽ được giữ lại khi chuyển scene. Đối tượng này vẫn có thể được giải phóng bằng hàm Destroy.

*\* Đối với các GameObject của scene, hàm chỉ có tác dụng nếu GameObject đó không là con của bất kì đối tượng nào, toàn bộ cấu trúc và các đối tượng con của nó sẽ được giữ lại qua mọi thao tác chuyển scene.*

1. LẬP TRÌNH GAME UNITY

Game Engine (Công cụ game) là một phần mềm dùng để xây dựng và thiết kế các trò chơi điện tử. Các nhà lập trình game phải kết hợp các ngôn ngữ lập trình và các công cụ game phù hợp để có thể tạo ra một sản phẩm game hoàn chỉnh.

Hiện nay, trên thế giới có rất nhiều công cụ game nổi tiếng như Unreal Engine, CryEngine, id Tech, Source, Godot,.. Trong số đó, không thể kể đến cái tên nổi bật Unity đã và đang chiếm ưu thế khá lớn trong ngành lập trình Game hiện nay.

* 1. Tổng quan về Unity

Unity là một công cụ trò chơi đa nền tảng được phát triển bởi Unity Technologies, chủ yếu được sử dụng để phát triển trò chơi điện tử và mô phỏng cho máy tính, bảng điều khiển và thiết bị di động.

Với các tính năng đa nền tảng, Unity phổ biến với cả các nhà phát triển game tự do và trong các studio game. Nó được sử dụng để tạo các trò chơi như Pokemon Go, Heathstone, Rimworld, Cuphead và nhiều trò chơi khác.

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#.

Bởi vì sự có mặt lâu từ năm 2005, Unity đã phát triển một lượng lớn người dùng và một thư viện tài nguyên khổng lồ. Unity không chỉ có tài liệu tuyệt vời, mà còn có vô số video và hướng dẫn trực tuyến đáng kinh ngạc cho người dùng.

Chính vì vậy, Unity là một sự lựa chọn thông minh cho người mới bắt đầu tiếp cận với các công cụ game. Nằm trong danh sách các công cụ trò chơi điện tử , Unity đóng vai trò như một cổng thông tin kiến ​​thức và tài nguyên được xây dựng chỉ dựa trên cộng đồng rộng lớn của họ.

**Ưu điểm khi sử dụng Unity**

Đối với người mới bắt đầu, Unity là một công cụ toàn diện tốt có thể đáp ứng hầu hết các nhu cầu của bất kỳ ai. Unity cũng tuyệt vời cho các trò chơi tạo mẫu.

Lý do chính để chọn Unity là thư viện tài nguyên khổng lồ dành cho mọi người. Ngay cả những nhà phát triển có kinh nghiệm cũng có thể tiết kiệm thời gian và học hỏi được nhiều điều từ cộng đồng.

Unity cũng cung cấp một bộ công cụ dựa trên điện toán đám mây (Cloud Computing) mạnh mẽ để dễ dàng kiếm tiền từ trò chơi và thêm khả năng nhiều người chơi.

Với Unity Analytics , Unity Ads, Unity Collaborate và Unity Multiplayer, người dùng có quyền truy cập vào một bộ công cụ để tạo các trò chơi động (dynamic games). Rất ít công cụ trò chơi khác cung cấp nhiều tính năng tập trung này.

Unity rất linh hoạt và có thể được sử dụng để tạo bất kỳ loại trò chơi điện tử nào, vì vậy đây là một nền tảng tuyệt vời cho các nhà lập trình game mới bắt đầu hoặc các nhà phát triển muốn thử xây dựng trò chơi mẫu cho mình.

* 1. Các thành phần cơ bản trong Unity
     1. Project Wizard and Project Panel

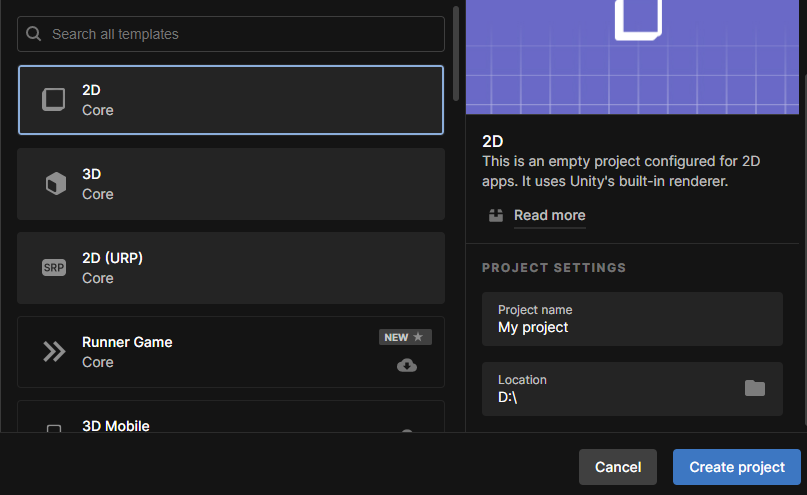
Project: trong Unity nó là một ứng dụng cơ bản. Trong thực tế nó có nghĩa là mỗi khi làm một game mới, sẽ cần một Project mới.

Trong Unity 1 Project = một Game. Một Project đơn giản chỉ là nơi chứa các files và data. Để tạo một Project trong unity, click New Project nó sẽ hiển thị view để nhập thông tin.



Hình : Tạo Project

Sau đó, chọn loại project muốn tạo ví dụ như: 2D hoặc 3D. Nhập tên project và chọn nơi lưu trữ project. Cuối cùng là click vào Create project.



Hình : Tạo project

Sau khi tạo xọng Unity sẽ hiển thị giao diện mặc định. Đây là nơi dành phần lớn thời gian để hoàn thành một game.

* Thanh Menu
* Game: hiển thị thực tế game trên màn hình khi run game.
* Scene: là nơi làm việc chính. Đây chính là vùng làm việc kéo thả thiết kế các đối tượng trong game.
* Hierarchy : list các đối tượng trong Scene.
* Project : hiên thị các folder, file trong project.
* Console: hiển thị lỗi trong projects.

*\*Có thể thay đổi vị trí, kích thước các vùng này bằng cách kéo thả, co dãn theo ý muốn.*

Trong Project folder thường bao gồm 4 thư mục con: Library, Assest, ProjectSettings, và Temp. Project Panel trong Unity interface sẽ hiển thị các nội dung của thư mục Assest, vì đây là nơi các file Game sẽ được lưu trữ khi import. Ba thư mục còn lại có chứa data phức tạp, và được quản lý độc quyền bởi Unity. Nó khuyến khích không bao giờ để tự loại bỏ hoặc chỉnh sửa các file từ bất cứ nơi nào bên trong Project folder hoặc các folder con của nó. làm như vậy có thể dẫn đến Project bị hỏng nếu các file không được sao chép một cách thích hợp.

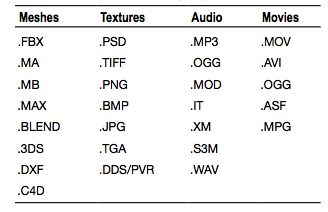
* + 1. Assest and Project File

Assest là nơi chứa tất cả các file trong Game như: Meshes, Textures, Movies, Animations, Sounds, Music, Text data ...... Các folder, file tự tạo ra theo cách của riêng người lập trình làm sao để dễ dàng quản lý nhất, nên đắt tên theo các chức năng của chúng. Khi muốn đặt một mesh hoặc object hay file bên trong Game, sẽ cần phải import lần đầu tiên như là một tài sản của các dự án. Chỉ có tài sản trong Bảng dự án có thể được đưa vào trò chơi. Có thể nhập khẩu Assest vào các Project sử dụng một trong hai phương pháp.

* Có thể chọn Asset ➤ import New Asset từ menu ứng dụng (xem hình dưới).
* Có thể kéo và thả file từ Windows Explorer hay Finder trực tiếp vào Bảng điều chỉnh dự án.

Các phương pháp trước đây cho phép chỉ nhập một file tại một thời gian, trong khi sau này cho phép nhiều file được inport với nhau.

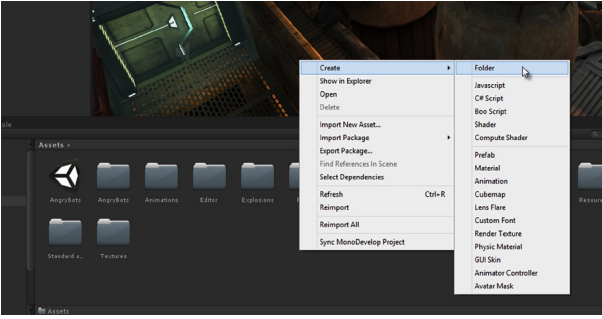
Một Assest khi đã import, nó sẽ xuất hiện trong cửa sổ Project, Unity chỉ chấp nhận các file có các định dạng sau:



Hình : Định dạng file

*\*Tất cả các meshes không exported trực tiếp sang định dạng FBX (như MA hoặc BLEND) yêu cầu 3D phù hợp phần mềm mô hình phải được cài đặt trên hệ thống trong import vào Unity. Unity nội bộ sử dụng phần mềm này export các định dạng nguồn gốc FBX.*

Cả 2D và 3D game thường sử dụng rất nhiều assets - perhaps có lẽ hàng trăm. Trò chơi 2D dựa vào nhất là về textures và materical. Đôi khi tổng kích thước của chúng có thể hàng GB. Nói về Assest nói chung, đó là thực tế tốt để tổ chức Assest trong của sổ Project. Điều này đảm bảo có thể tìm thấy assest một cách nhanh chóng khi cần chúng. textures thường được bố trí trong một folder, meshes ở một folder khác, sounds cũng trong một folder khác, và cứ như vậy. Tránh trộn lẫn các loại với nhau. Để tạo folder trực tiếp từ cửa sổ Project, kích chuột phải vào trong và chọn Create ➤ Folder từ menu Scene. Hoặc, Assest ➤ Create ➤ Folder từ menu ứng dụng.

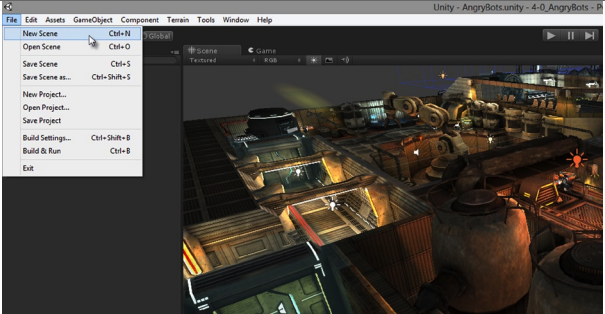


Hình : Tạo Folder

Các Assests được import vào Unity được tự động gán một loạt các thiết lập và thuộc tính trong quá trình nhập khẩu bởi các công cụ Unity. Những kiểm soát Assest làm việc khi bao gồm trong Game. Có thể xem các thiết lập và thay đổi chúng một cách dễ dàng. Chỉ cần chọn Assest trong cửa sổ Project và điều chỉnh các thiết lập của nó, xuất hiện trong Inspector Object. chắc chắn nhấn vào nút Apply ở phía dưới cùng của thanh tra khi đang thực hiện để xác nhận và lưu thay đổi cho Assest được chọn.

* + 1. Scene

Khi đã tạo một Project và import tất cả các Assest, nó thường là thời gian để bắt đầu làm việc levels trong Game. Trong Unity, level (hoặc một màn chởi) được gọi là một Scene. 1 scene = 1 level. Một Game nhiều mức, và vì vậy hầu hết các Project đều có nhiều Scene. Unity chủ yếu là một Engine 3D, và điều này có nghĩa là loại duy nhất của Scene có sẵn là scene 3D. Một scene 3D có không gian phối hợp 3D , với trục X và Y, Z. Để tạo ra mức độ 2D trong Unity, chúng tôi chỉ đơn giản tạo cảnh 3D chuẩn và sau đó cấu hình chúng để hành xử như 2D. Người chơi sẽ có thể không bao giờ biết sự khác biệt trừ khi họ là develop Unity pro! Để tạo ra một new Scene, chọn File ➤ New Scene từ trình đơn ứng dụng, hoặc nhấn phím tắt Ctrl + N. Sau khi tạo một Scene, hãy lưu nó bằng cách chọn File Save ➤ Scene. Nó sẽ được lưu trong Assest folder. Có thể một folder Scenes và kéo chúng vào đó.



Hình : Tạo Scene

**Điều hướng Cảnh (Scene) và Chế độ xem (Viewports)**

Có thể mở và xem xét bất cứ Scene nào trong Project bằng cách Click đúp vào nó từ cửa sổ Project. Mỗi khi tạo ra một Scene mới, nó sẽ tự động được mở ra. Khi mở một scene trong trình Unity editor, có thể xây dựng levels và kiểm tra chúng. Các thành phần Viewport của interface cho phép nhìn thấy bên trong scene. Scene trong Hierarchy cũng liệt kê tất cả các Object theo tên: tất cả các lights, cameras, environments, creatures, power-ups, và các đối tượng khác được liệt kê ở đây, ngay cả các đối tượng không thể nhìn thấy. Danh sách này là đầy đủ; nếu một object sống trong scene hoạt động, sau đó nó được bao gồm trong cửa sổ Hierarchy. Bằng cách nhấp đúp chuột vào một object trong danh sách đó, không chỉ chọn các object trong scene nhưng canh giữa Viewports vào nó cho một cái nhìn tốt hơn. Hơn nữa, các thuộc tính cho các object được chọn được thể hiện trong cửa sổ Inspector.

*\*Các khu vực Viewport được chia trên hai cửa sổ, Scene và Game. Cửa sổ Scene là một khung nhình giám sát Scene. Unity sử dụng các cửa sổ này để xây dựng lên những Scene. Cửa sổ Game hiển thị cùng một Scene nhưng từ quan điểm của các player chứ không phải là các developer. Nó cho chúng ta Scene như nào khi chơi. Chúng ta sẽ làm việc trong cả các cửa sổ này cho các trò chơi 2D.*

* 1. Một số bước lập trình game 2D với Unity
     1. Các nguyên tắc cơ bản

**GameObjects:** Là các đối tượng cơ bản trong Unity đại diện cho các nhân vật, đạo cụ, phong cảnh, v.v. Mọi đối tượng trong trò chơi là một GameObject.

GameObjects đại diện cho các mục trong trò chơi; không gian mà đặt chúng để xây dựng cấp độ được gọi là bối cảnh. Khi lập trình Game 2D, có thể bỏ qua trục thứ 3 (trục z).

Hành vi của GameObjects được xác định bởi các khối chức năng được gọi là các thành phần. Các thành phần sau là cơ bản cho trò chơi 2D:

**Transform**: thành phần Transform sẽ xác định Vị trí, Xoay và Tỷ lệ của từng GameObject trong cảnh. Mỗi GameObject đều có một thành phần Transform.

**Sprite Renderer**: thành phần Sprite Renderer kết xuất Sprite và điều khiển nó trông như thế nào trong một cảnh.

**Máy ảnh**: thiết bị thu và hiển thị thế giới cho người chơi. Đánh dấu Máy ảnh là Orthographic sẽ xóa tất cả góc nhìn khỏi chế độ xem của Máy ảnh. Điều này chủ yếu hữu ích cho việc tạo các trò chơi isometric hoặc 2D.

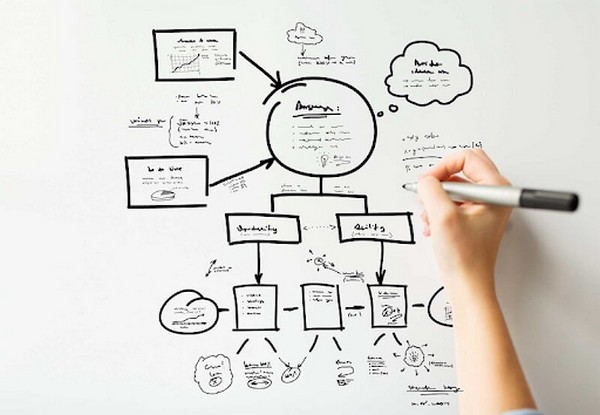
**Collider 2D**: thành phần này xác định hình dạng của GameObject 2D cho các mục đích va chạm vật lý

Các thành phần là các đại diện giao diện người dùng của các lớp C #; có thể sử dụng các tập lệnh để thay đổi và tương tác với các thành phần hoặc tạo các tập lệnh mới.

* + 1. Viết kịch bản Game

Tất cả các trò chơi 2D đều cần kịch bản. Các kịch bản phản hồi thông tin đầu vào từ người chơi và sắp xếp các sự kiện trong trò chơi xảy ra khi cần thiết.

Kịch bản được đính kèm với GameObjects và bất kỳ kịch bản nào tạo ra đều kế thừa từ lớp MonoBehaviour.



Hình : Viết kịch bản game

* + 1. Sprites

Sprites là các đối tượng đồ họa 2D. Sử dụng Sprites cho tất cả các loại trò chơi 2D. Ví dụ: có thể nhập hình ảnh của nhân vật chính dưới dạng Sprite.



Hình : Ví dụ về đồ họa 2D trong Unity

Có thể sử dụng một bộ sưu tập Sprites để xây dựng một nhân vật. Điều này cho phép kiểm soát tốt hơn chuyển động và hoạt ảnh của các nhân vật.

* + 1. Xây dựng môi trường trong trò chơi

Thiết kế môi trường đề cập đến quá trình xây dựng các cấp độ và môi trường trong trò chơi. Có thể kết hợp các công cụ thiết kế môi trường trong phần này theo bất kỳ cách nào phù hợp nhất với trò chơi.

* + 1. Hoạt hình nhân vật

Có ba cách khác nhau để có thể tạo hoạt ảnh cho các nhân vật 2D:

|  |  |
| --- | --- |
| **Loại hoạt hình 2D** | **Mục đích** |
| **Frame-by-frame** | Lý do nghệ thuật, nếu muốn trò chơi của mình có phong cách nghệ thuật hoạt hình cổ điển. Hoạt ảnh từng khung hình tương đối tốn nhiều tài nguyên, cả để tạo và chạy. |
| **Cutout** | Hình ảnh động khung xương mượt mà, khi các nhân vật không yêu cầu khớp nối thực tế. |
| **Skeletal** | Hoạt ảnh khung xương mượt mà trong đó Sprites uốn cong theo cấu trúc xương. Sử dụng điều này khi các nhân vật cần một cảm giác hữu cơ hơn. |

* + 1. Đồ họa

Phần này mô tả các tùy chọn đồ họa khi sử dụng UniversalRender Pipeline (URP) – Một đường dẫn kết xuất thực hiện một loạt các hoạt động lấy nội dung của Bối cảnh và hiển thị chúng trên màn hình.

* + 1. Vật lý 2D

Cài đặt Vật lý 2D xác định giới hạn về độ chính xác của mô phỏng vật lý trong trò chơi 2D.

* + 1. Âm thanh

Có thể thêm nhạc nền và hiệu ứng âm thanh vào trò chơi trong Unity. Sử dụng phần mềm của bên thứ ba để tạo âm thanh và nhập nó vào Unity với các cài đặt được đề xuất.

* + 1. Giao diện người dùng

Nếu muốn thêm menu hoặc trợ giúp vào trò chơi, cần thiết lập giao diện người dùng. Để thiết lập giao diện người dùng, hãy sử dụng Unity UI

* + 1. Lập hồ sơ, tối ưu hóa và thử nghiệm

Việc lập hồ sơ cho phép xem các phần khác nhau trong trò chơi sử dụng nhiều tài nguyên như thế nào. Nên luôn lập hồ sơ trò chơi trên nền tảng phát hành mục tiêu của nó.

Sau khi lập hồ sơ, có thể sử dụng kết quả để cải thiện và tối ưu hóa hiệu suất.

Kiểm tra trò chơi và mã với Khung kiểm tra thống nhất.

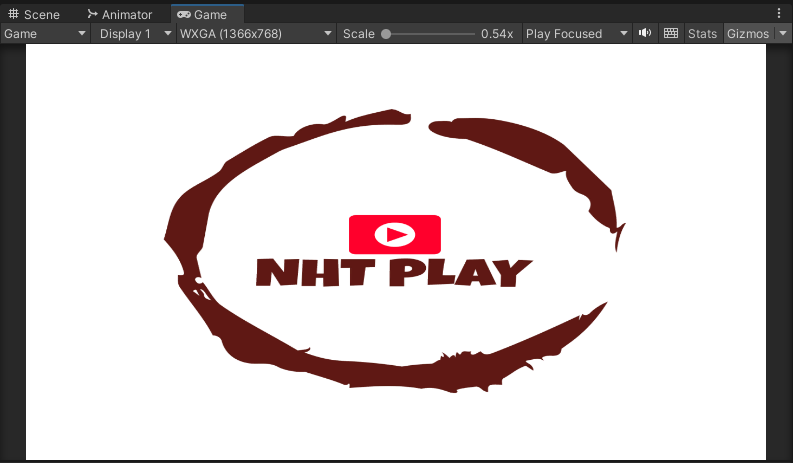
* + 1. Xuất bản

Khi đã hoàn thành, đã sẵn sàng xuất bản trò chơi.

Đây chỉ là các bước cơ bản để **lập trình game 2D bằng Unity.**

1. ỨNG DỤNG GAME 2D SWORD HUNTER
   1. Một số Scene trong Game

Khi bắt đầu dự án game xuất hiện logo của ứng dụng:



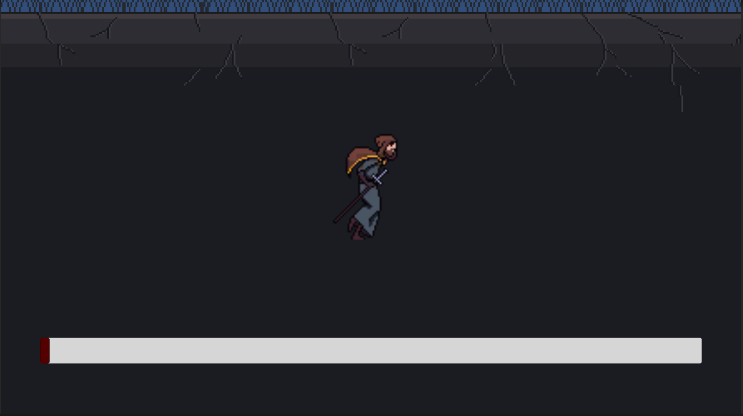
Hình : Màn hình Logo

Màn hình khi load game:

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Màn hình Loading



Hình : Màn hình Loading

Màn hình Menu xuất hiện sau khi loading, có thể chọn các màn chơi, chọn trang bị để sử dụng, cũng như mua các trang bị, hướng dẫn chơi và đóng ứng dụng.

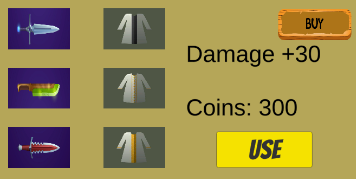
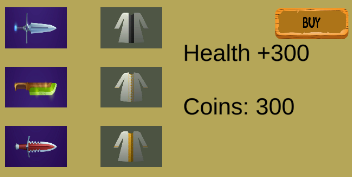
Ảnh có chứa Website

Mô tả được tạo tự động

Hình : Màn hình Menu

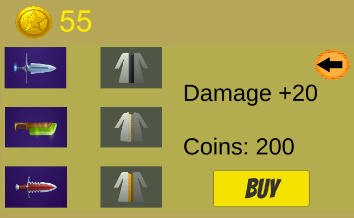
Một số trang bị trong game với các chỉ số và giá coins khác nhau. Trước khi mua trong Shop thì trang bị sẽ không hiện nút Use.

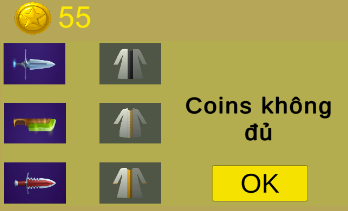
 

Hình : Một số trang bị trong game

Khi bấm vào nút Buy, sẽ xuất hiện Shop trang bị. Sau khi mua thành công sẽ quay lại màn hình Menu. Nếu coins không đủ xẽ xuất hiện thông báo.



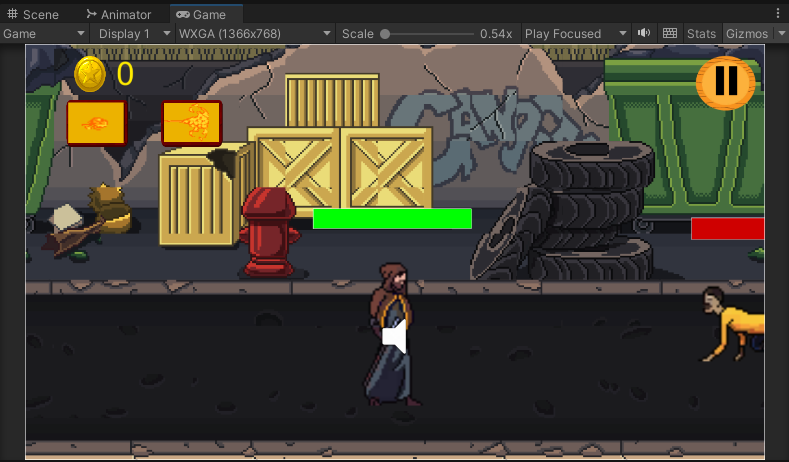
Hình : Shop trang bị



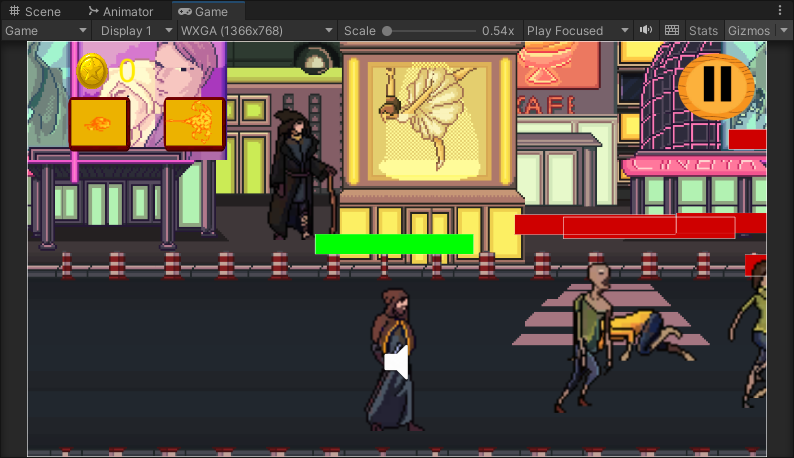
Hình : Shop trang bị

Nhấn Ok để quay lại Shop, muốn trở về kho trang bị thì nhấn vào mũi tên.

Một số màn hình khi chơi game:



Hình : Màn hình Map 1



Hình : Màn hình Map 2

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Màn hình Map 3

Màn hình Pause xuất hiện khi bấm vào nút Ảnh có chứa văn bản

Mô tả được tạo tự động trên góc màn hình. Nút bật âm thanh và tắt âm thanh trong game, nút resume và nút trở lại Menu.



Hình : Màn hình Pause

Màn hình Game Over xuất hiện khi nhân vật hết máu.



Hình :Màn hình Game Over

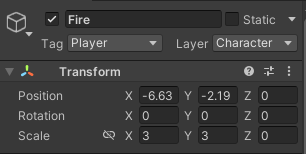
* 1. Nhân vật Player



Hình : Nhân vật Player

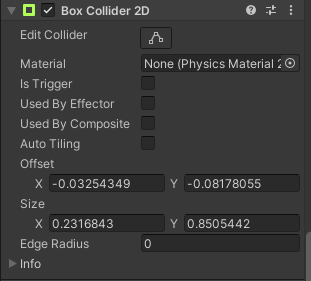
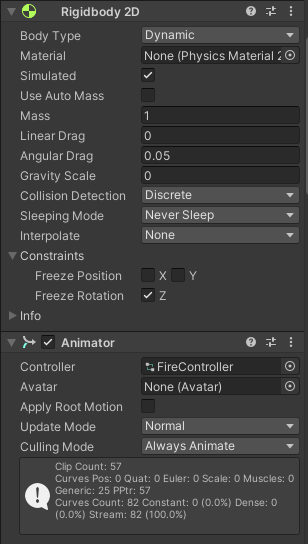
Khi tạo mới một nhân vật cần làm một số thao tác sau:

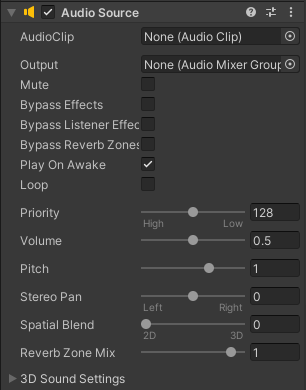
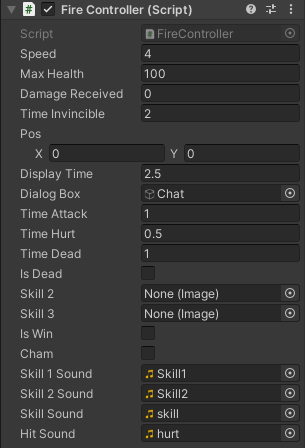
* Kéo thả hình của player vào trong Scene
* Tùy chỉnh kích thước và vị trí sao cho phù hợp
* Thêm các component như Box Collider 2D để nhận biết được sự va chạm giữa các phần trên map.
* Tạo Script để điều khiển player.
* Tạo Animation cho nhân vật chuyển động



Hình : Player

**Các Component có trong Inspector của Player**





Hình : Component Player

* + 1. Script

Script có tên là FireController.cs và được gắn vào player.

Cơ bản player sẽ có damage là 10, health là 100 và speed là 4.0f.

int health = 100;

int damage = 10;

float speed = 4.0f;

Dưới đây là một số phần cơ bản trong script:

**Skill**

Skill trong game và coins, hiển thị thời gian hồi skill và khi giết được quái vật, coins sẽ ngẫu nhiên trong một khoảng theo từng loại quái vật.

 Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Skiill và Coins

Skill thường: Đánh thường, thời gian hồi chiêu 1s, Key: Space.

Skill 1: Cầu lửa, thời gian hồi chiêu 2s, Key: Q.

Skill 2: Lửa bùng nổ, thời gian hồi chiêu 3s, Key: E.

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Skill 1: Đánh thường

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Skill 2: Cầu lửa

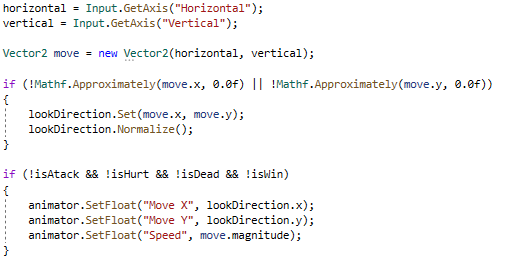
Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Skill 3: Lửa bùng nổ

**Moving**

Để nhân vật có thể di chuyển cần bắt sự kiện từ bàn phím, có nhiều cách để di chuyển nhân vật trong Unity tùy theo kịch bản game.



Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Moving Player

Khi nhân vật đang sử dụng skill hoặc khi nhân vật đang bị tấn công sẽ không thể di chuyển.

**Health**

Health sẽ hiển thị phía trên nhân vật. Khi quái bị quái vật tấn công máu của nhân vật sẽ bị thay đổi.

Ảnh có chứa văn bản

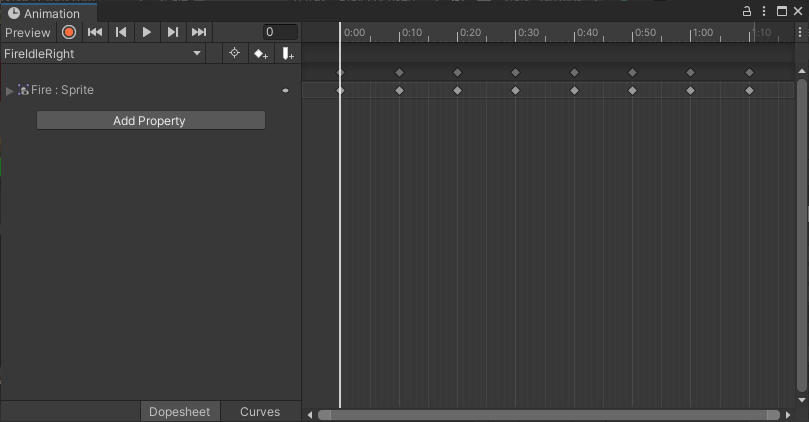
Mô tả được tạo tự động

Hình : Change Health

* + 1. Thay đổi trạng thái của nhân vật

**Đầu tiên, tạo một Animator Controller có tên là FireController và gán vào nhân vật.**

Chọn Window -> Animation -> Animation, xuất hiện cửa sổ Animation. Tạo các Animation Clip tương ứng theo từng hành động.



Hình : Animation

Trong Animator, chọn Parameters, click vào dấu +, để tạo các trạng thái của nhân vật.

Float: dùng để tạo Move x, Move y, Speed nhằm nhận biết được trạng thái khi nhân vật di chuyển.

Trigger: dùng để nhận biết các trạng thái khi bị quái vật tấn công, khi sử dụng skill và khi nhân vật chết.

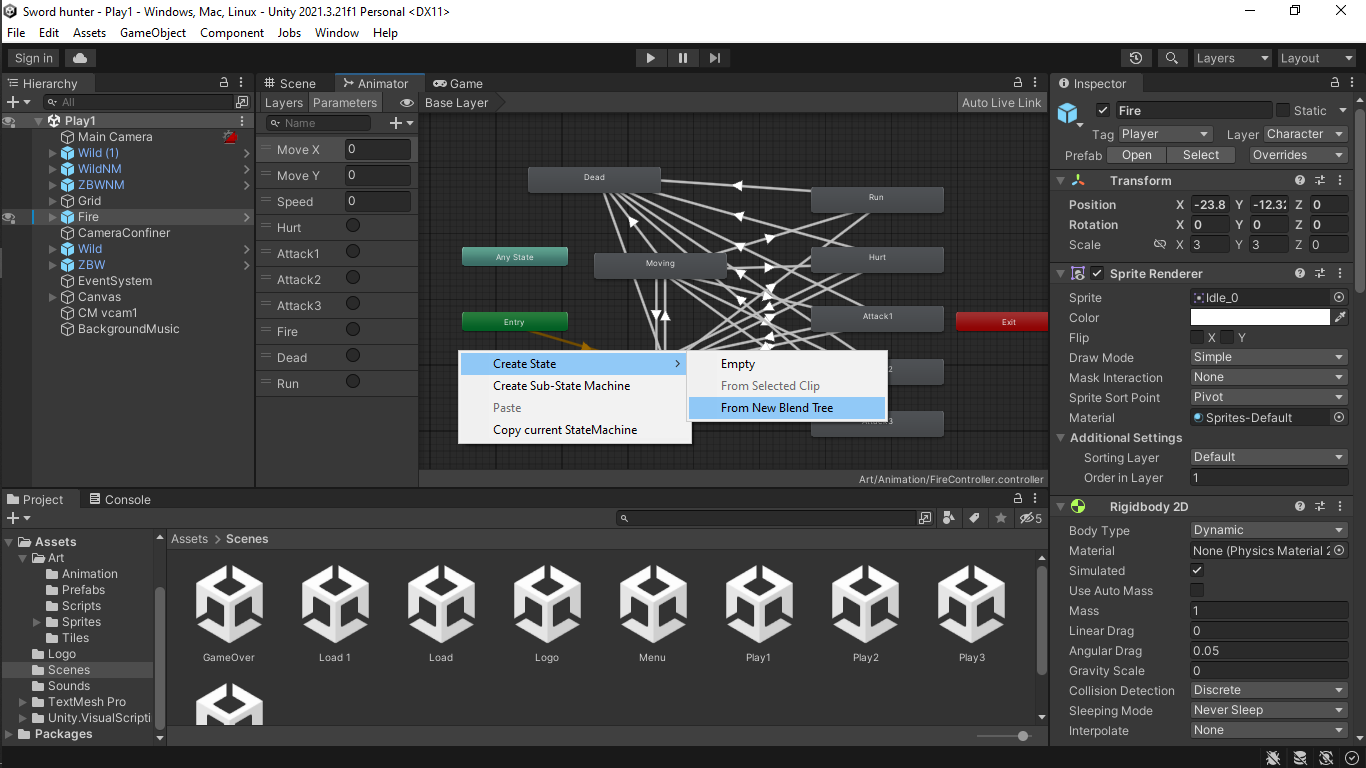
Ảnh có chứa văn bản, đồ điện tử, ảnh chụp màn hình, màn hình

Mô tả được tạo tự động

Hình : Parameters

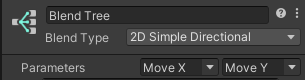
**Tiếp theo là tạo Blend Tree để kết hợp các animation clip phù hợp theo từng trạng thái.**

Đầu tiên, để tạo Blend Tree click phải chuột -> Create State -> From New Blend Tree.



Hình : Tạo Blend Tree

Click đúp vào Blend Tree, thay đổi type thành 2D Simple Directional, click chọn dấu + để thêm các Motion

 Ảnh có chứa văn bản, đồ điện tử, máy tính

Mô tả được tạo tự động

Hình : Blend Tree

**Kế tiếp, cần viết code trong Script để thay đổi trạng thái nhân vật.**

Khai báo đối tượng Animator để có thể thay đổi trạng thái nhân vật.

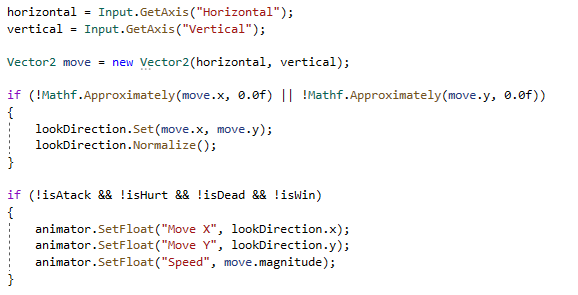
Animator animator;

Vector2 lookDirection = new Vector2(1, 0);

Trong void Start()

animator = GetComponent<Animator>();

Kế tiếp, khi nhấn phím di chuyển cần thêm câu lệnh chuyển trạng thái tương ứng.



Hình : Script Animation

Như vậy, bằng đối tượng Animator, chúng ta có thể thay đổi trạng thái nhân vật, kết hợp với di chuyển nhân vậy, từ đó xây dựng được nhân vật game di chuyển kèm Animation trùng khớp.

* 1. Một số thành phần trong game
     1. NPC

Ảnh có chứa văn bản, bầu trời, biển hiệu

Mô tả được tạo tự động Ảnh có chứa văn bản, bầu trời, biển hiệu

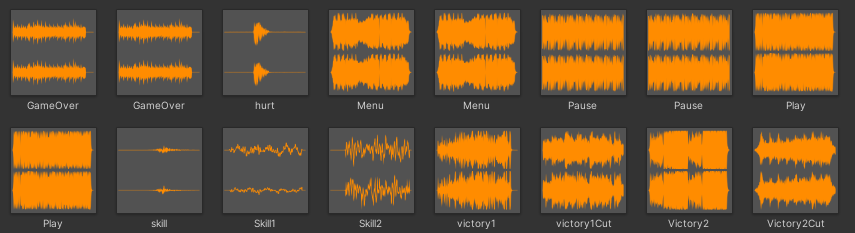
Mô tả được tạo tự động Ảnh có chứa văn bản, biển hiệu, bầu trời

Mô tả được tạo tự động

Hình : Một số NPC

Các NPC sẽ xuất hiện với một lời nhắc nhở khi bắt đầu màn chơi. Lời nhắc nhở về sự nguy hiểm trong từng màn chơi sẽ tăng lên.

* + 1. Sound

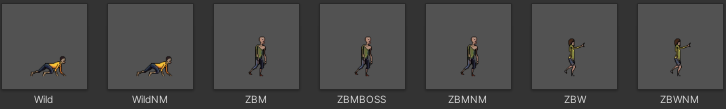


Hình : Sound

Sound là một thành phần không thể thiếu trong các trò chơi ví dụ như âm thanh trong màn chơi, âm thanh khi sử dụng skill hay âm thanh khi bị tấn công,..

* + 1. Quái vật

Các quái vật khác nhau sẽ có một số sự khác nhau về damage, health, speed, … Nhưng nhìn chung có một số sự tương đồng sau:



Hình : Quái và Boss

Quái vật sẽ di chuyển từ trái sang phải theo thời gian. Khi bị tấn công quái vật sẽ di chuyển với tốc độ gấp đôi và đuổi theo player. Riêng boss sau khi xuất hiện sẽ di chuyển về phía player.

Ảnh có chứa văn bản

Mô tả được tạo tự động Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Enemy Moving

Khi tấn công quái vật sẽ ngẫu nhiên 1 trong 2 skill.

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình : Enemy Skill

Trên đây là một số thành phần cơ bản trong game. Để hoàn thành được ứng dụng game này cần rất nhiều thành phần lớn nhỏ khác như tạo map, các canvas,…

KẾT LUẬN VÀ KIẾN NGHỊ

Thông qua quá trình xây dựng ứng dụng game 2D Sword Hunter bằng Unity, em đã vận dụng các kiến thức đã được học trên trường và kiến thức đã học được trong quá trình hoàn thiện ứng dụng. Qua đó, em rút ra được nhiều kinh nghiệm cho bản thân mình hơn, chuyên môn cũng được cải thiện hơn. Bên cạnh các kiến thức chuyên môn, em còn học hỏi được nhiều kỹ năng làm việc như: Cách làm việc cá nhân, cách lên kế hoạch cho công việc, cách quản lý thời gian,… Unity là một công cụ rất hữu ích trong việc lập trình game, Unity đóng vai trò như một cổng thông tin kiến ​​thức và tài nguyên được xây dựng. Mặc dù còn gặp một số khó khăn trong quá trình hoàn thiện sản phẩm, nhưng em đã cố gắng rất nhiều để thực hiện đề tài một cách tốt nhất. Em nghĩ bản thân cần cố gắng nỗ lực hơn nữa trong tương lai để hoàn thiện bản thân.

TÀI LIỆU THAM KHẢO

**Website tham khảo**

[1] RYE NGUYEN. (2020, September 25). *GameObject - Thao Tác với C# Script — Lập Trình Game*. (n.d.-b). IO Stream. https://www.iostream.vn/lap-trinh-game/gameobject-thao-tac-voi-c-script-y11g3U

[2] Hùng, & Hùng. (2022, April 25). Lập trình game Unity là gì? Cách biến ý tưởng thành sản phẩm Game. *VTC Academy | Học viện Công nghệ thông tin và Thiết kế VTC (VTC Academy)*. https://vtc.edu.vn/lap-trinh-game-unity/

[3] Cường T. M. (2022, February 24). *Cơ bản về lập trình game Unity 2D - Phần 1*. techmaster.vn. https://techmaster.vn/posts/33356/hoc-lap-trinh-game-unity-co-ban

[4] Anh, M. (2022, February 24). *Các câu lệnh cơ bản với Script trong lập trình Game Unity - Cao Đẳng FPT Polytechnic xét tuyển*. Cao Đẳng FPT Polytechnic Xét Tuyển. https://caodang.fpt.edu.vn/tin-tuc-poly/cac-cau-lenh-co-ban-voi-script-trong-lap-trinh-game-unity.html