

MỤC LỤC

DANH SÁCH MỘT KÝ HIỆU SỬ DỤNG TRONG ĐỀ TÀI.....	2
I. GIỚI THIỆU CHUNG	3
1. Mục tiêu TTCS	3
2. Mục tiêu và định hướng cá nhân	3
II. NỘI DUNG THỰC TẬP.....	4
1. Nội dung thực tập đề xuất	4
2. Nội dung học tập và báo cáo tiến độ từng tuần	4
III. NỘI DUNG DỰ ÁN TTCS.....	6
Chương 1. Tổng quan đề tài.....	6
1.1. Lý do chọn đề tài.....	6
1.2. Giới hạn và phạm vi đề tài	6
1.3. Nội dung thực hiện.....	6
1.4. Phương pháp tiếp cận.....	6
Chương 2. Cơ sở lý thuyết.....	8
2.1. Tổng quan về Unity.....	8
2.1.1. Đối tượng tham gia hệ thống	8
2.1.2. Lịch sử của Unity	8
2.2. Tổng quan về các thành phần trong Unity	9
2.2.1. Assets.....	9
2.2.2. Scenes.....	10
2.2.3. Game Object	10
2.2.4. Components.....	11
2.2.5. Scripts	11
2.2.6. Collider	12
2.2.7. Rigidbody	13
2.2.8. Prefabs	13
2.2.9. Sprite.....	14
2.2.10. Animator.....	14
2.2.11. Audio Source.....	15
2.2.12. Camera	16
2.2.13. Transform	17
2.2.14. Renderer	17
2.3. Nguyên tắc thiết kế.....	18

2.3.1. Nguyên tắc 1: Thiết kế giao diện game chặt chẽ và dễ sử dụng.....	18
2.3.2. Nguyên tắc 2: Game phải được sử dụng một cách mượt mà	18
2.3.3. Nguyên tắc 3: Cách cài đặt game phải dễ dàng	18
Chương 3. Nội dung thực hiện	19
3.1. Phân tích đề tài.....	19
3.1.1. Tóm tắt.....	19
3.1.2. Phân tích thị trường	19
3.1.3. Cơ chế gameplay	22
3.2. Xác định yêu cầu	22
3.2.1. Giao tiếp hệ thống.....	22
3.2.2. Giao tiếp điều khiển.....	23
3.2.1. Giao tiếp giao diện	23
3.3. Kịch bản game.....	23
3.4. Xây dựng game.....	23
3.4.1. Xây dựng hệ thống Controller cho Player	23
3.4.2. Xây dựng hệ thống Basic Combat cho Player	29
3.4.3. Xây dựng camera theo dõi Player	31
3.4.4. Xây dựng AI Enemy MeleeAttack và sơ đồ Animator	32
3.4.5. Xây dựng AI Enemy RangedAttack và sơ đồ Animator	34
3.4.6. Xây dựng các chương ngại vật	35
3.4.7. Xây dựng thông báo	36
3.4.8. Xây dựng các phân cảnh và chuyển cảnh	36
3.4.8. Xây dựng hệ thống âm thanh	37
3.5. Demo Game	38
3.5.1. Giao diện màn hình chờ Menu	38
3.5.2. Màn hình Pause Game	39
3.5.4. Màn hình Game Play.....	40
IV. KẾT LUẬN.....	42
1. Kết quả đạt được của đề tài	42
2. Hạn chế của đề tài	42
3. Hướng phát triển của đề tài	42
TÀI LIỆU THAM KHẢO.....	44

DANH SÁCH MỘT SỐ KÝ HIỆU SỬ DỤNG TRONG ĐỀ TÀI

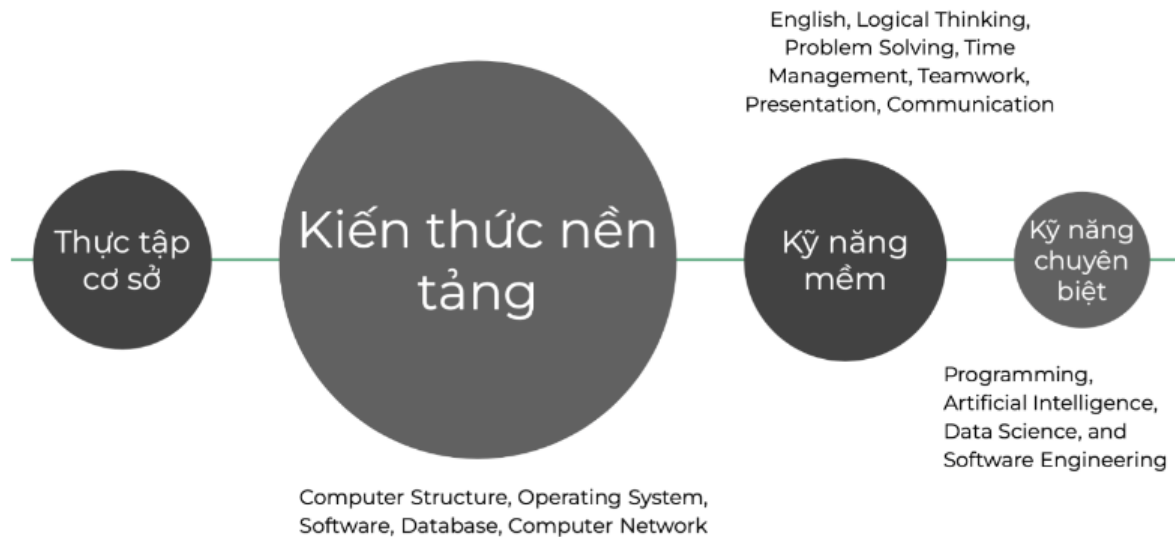
Từ viết tắt	Từ đầy đủ	Ý Nghĩa
TTCS	Thực tập cơ sở	Là kì kiến tập/học tập thực tế tại doanh nghiệp/cơ quan/tổ chức, mang tính chất tiếp cận với nghề nghiệp, công việc liên quan ngành nghề đang theo học.
Player	Người chơi	Là người thực hiện nhân vật chính trong trò chơi.
Controller	Điều khiển	Hệ thống điều khiển.
Basic	Cơ bản	Để miêu tả các đối tượng ở trạng thái cơ bản nhất.
Combat	Đánh	Là hệ thống tấn công.
Enemy	Kẻ thù	Là những đối tượng tương tác với nhân vật theo hoàn cảnh cụ thể.
MeleeAttack	Cận chiến	Tương tác với phạm vi gần.
RangedAttack	Tầm xa	Tương tác với phạm vi xa hơn.
Trap	Cạm bẫy	Là những đối tượng tương tác với nhân vật theo hoàn cảnh cụ thể.
Scene	Cảnh	Phân cảnh của game.
Sound	Âm thanh	Âm thanh trong game.

I. GIỚI THIỆU CHUNG

1. Mục tiêu TTCS

Mục tiêu thực tập cơ sở là củng cố kiến thức cơ sở cho sinh viên khi vào phần chuyên ngành tạo tiền đề cho các sinh viên trong kỳ thực tập năm tiếp theo.

Kiến thức và Kỹ năng chuyên môn



Yêu cầu cho mỗi sinh viên cần được bao phủ tối thiểu 2/4 mảng kiến thức, trong đó mảng kiến thức về lập trình là bắt buộc. Sinh viên lựa chọn một project hoặc nhóm các bài tập theo từng mảng kiến thức.

Mỗi 2 tuần, các nhóm hay cá nhân cần có báo cáo tiến độ cụ thể các việc thực hiện được trong tuần. Cuối kỳ sẽ đánh giá dựa trên kết quả thực hiện được của project và cả quá trình học hỏi. Các nhóm hay cá nhân sẽ trình bày lại và gửi kết quả báo cáo. Điểm sẽ được đánh giá dựa trên báo cáo tiến độ và trình bày project vào cuối kỳ.

2. Mục tiêu và định hướng cá nhân

- Mục tiêu: Trong tương lai gần có thể xin thực tập tại vị trí Game Dev.
- Định hướng cá nhân: Phát triển bản thân theo hướng Game Dev.

II. NỘI DUNG THỰC TẬP

1. Nội dung thực tập đề xuất

- Đề xuất đề tài: Lập trình Game 2D Platform cho PC
- Phần mềm sử dụng: Unity, Visual Studio
- Ngôn ngữ lập trình: C#

2. Nội dung học tập và báo cáo tiến độ từng tuần

STT	Topics	Weeks	Target
1	Học những kiến thức cơ bản về Unity và C#	1+2	- Với Player: + Animation với player + Basic Controller như Idle, Move, Jump, SlideWall và 1 số chức năng bổ sung trong quá trình chơi như Dash, Bash + Basic Combat - Với Enemy: + basic enemy - Với Scene: + Xây dựng map đơn giản với Grid
2	Tìm hiểu về sơ đồ và cách thức hoạt động của AI enemy	3+4	- Sơ đồ và script AI enemy với 2 dạng enemy cận chiến và tầm xa - Đang bổ sung thêm 1 số Ability cho Player
3	Tìm hiểu về từng chức năng của Unity như Prefab, Animator, Slice Sprites...	5+6	- Hoàn thành chương ngại vật - Cải thiện Controller, Ability và các chỉ số cho Player - Hoàn thành giao diện thanh máu
4	Tìm hiểu về những chức năng nâng cao như Canvas, Shader để tạo hiệu ứng cho cảnh và xây dựng map.	7+8	- Game Menu - Pause Game - Respawn và CheckPoint - Xây dựng đồ họa cho Background - Xây dựng hiệu ứng cho Background + Xây dựng map
5	Tìm hiểu về hiệu ứng	9+10	- Âm thanh - Lời thoại

	chuyển cảnh, lời thoại và âm thanh		- Chuyển cảnh => Hoàn thành Gameplay
--	--	--	---

III. NỘI DUNG DỰ ÁN TTCS

Chương 1. Tổng quan đề tài

1.1. Lý do chọn đề tài

Trong những năm gần đây, khoa học và kỹ thuật phát triển mạnh mẽ, công nghệ cũng có những bước tiến vượt bậc đặc biệt là về mảng điện thoại di động. Điện thoại di động là thiết bị tiện ích, dễ sử dụng, nhỏ gọn, có thể sử dụng mọi lúc, mọi nơi.... Các App dần dần xuất hiện và ngày càng phát triển hơn về cả số lượng và chất lượng.

Các ứng dụng điện thoại xuất hiện nhằm phục vụ nhu cầu tất yếu của người dùng như: chơi game, nghe nhạc, chụp ảnh, quay video, xem phim... Nhu cầu của người dùng càng cao, App xuất hiện càng nhiều. Tuy nhiên, không phải App nào được làm ra cũng có chất lượng tốt.

Gần đây nhu cầu giải trí của người dùng ngày càng cao ai cũng muốn giải trí giải tỏa tinh thần sau những ngày làm việc mệt nhọc. Vì thế họ tìm đến các hoạt động giải trí ngoài trời nhằm thư giãn đầu óc. Tuy nhiên không phải ai cũng có thời gian, cơ hội để mà tham gia các hoạt động giải trí ngoài trời hoặc là họ không thích đi xa. Chính vì thế mà họ tìm đến thú vui bằng các game ngay trên điện thoại di động của mình. Do biết được nhu cầu của người dùng mà tôi tham gia xây dựng một ứng dụng game 2D Platform trên Unity nhằm phục vụ nhu cầu giải trí của những người sử dụng.

1.2. Giới hạn và phạm vi đề tài

Đề tài xây dựng game 2D Platform sử dụng:

- Công cụ: Unity
- Công cụ lập trình: Visual Studio
- Ngôn ngữ lập trình: C#

1.3. Nội dung thực hiện

Nội dung thực hiện cụ thể:

- Thiết kế đặc tả hệ thống quản lý rõ ràng, nhất quán.
- Xây dựng chắc chắn cơ bản của game.
- Thiết kế đặc tả hệ thống
- Kiểm thử hệ thống.
- Triển khai thực nghiệm hệ thống máy tính.

1.4. Phương pháp tiếp cận

- Cách tiếp cận: Trực tiếp tham gia sử dụng các ứng dụng game đã được xây dựng để rút ra các kinh nghiệm để xây dựng game của mình được tốt hơn, chế

độ phục vụ tốt hơn. Bắt tay trực tiếp xây dựng game, chưa cần phải có kiến thức nhưng sẽ tạo ra được lối tư duy đúng đắn khi lập trình game.

- Sử dụng các phương pháp nghiên cứu: đọc tài liệu trên internet và từ nhà sản xuất Unity.

Chương 2. Cơ sở lý thuyết

2.1. Tổng quan về Unity

2.1.1. Đối tượng tham gia hệ thống

Đã qua thời kỳ làm game trên nền Flash căn bản và buồn chán với những chuyển động cứng nhắc. Unity mang đến sức mạnh kỳ diệu cho nhân vật mà chúng ta muốn thể hiện sống động hơn trong không gian ba chiều đầy huyền ảo. Công nghệ cao này tạo ra một bước đột phá mới về sự khác biệt trong công nghệ làm game hiện nay, mang đến cho người chơi 1 cảm giác rất khác lạ và hào hứng trong từng chuyển động, tương lai công nghệ này được áp dụng vào game Việt Nam sẽ mở ra một trang mới trong thế giới game 2D, 3D huyền ảo.

Unity được dùng để làm video game, hoặc những nội dung có tính tương tác như thể hiện kiến trúc, hoạt hình 2D, 3D thời gian thực. Unity hao hao với Director, Blender game engine, Virtools hay Unreal Engine trong khía cạnh dùng môi trường đồ họa tích hợp ở quá trình phát triển game là chính.

Unity là một trong những engine được giới làm game không chuyên cực kỳ ưa chuộng bởi khả năng tuyệt vời của nó là phát triển trò chơi đa nền. Trình biên tập có thể chạy trên Windows và Mac OS, và có thể xuất ra game cho Windows, Mac, Wii, iOS, Android. Game cũng có thể chơi trên trình duyệt web thông qua plugin Unity Web Player. Unity mới bổ sung khả năng xuất ra game trên widget cho Mac, và cả Xbox 360, PlayStation 3.

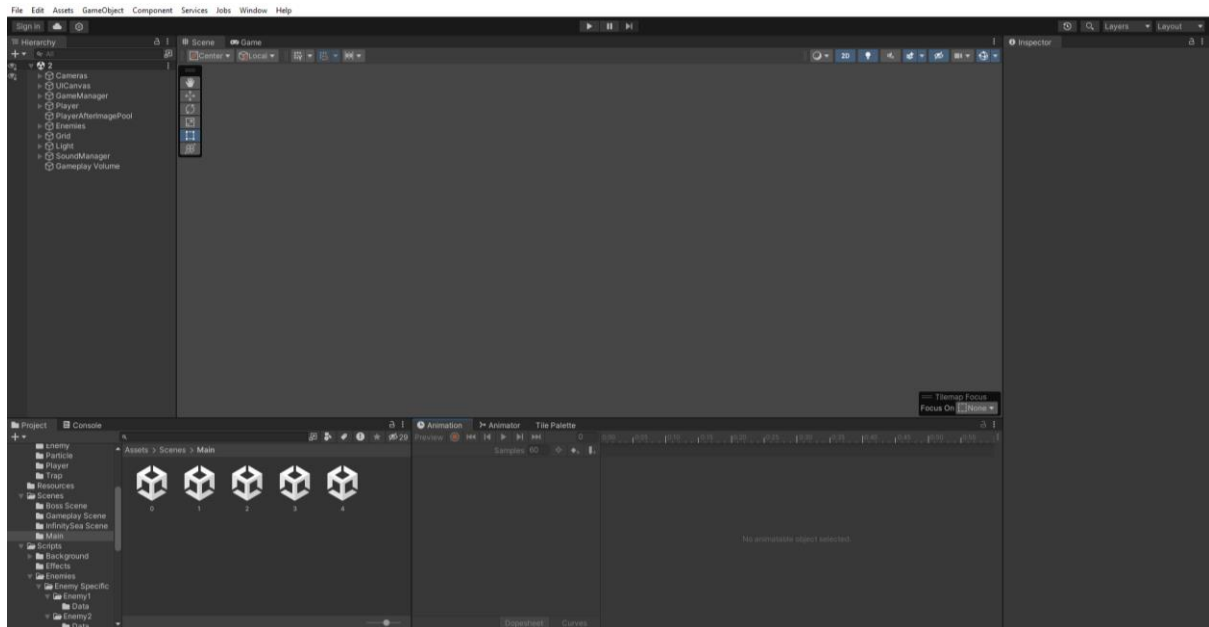
2.1.2. Lịch sử của Unity

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

Vào đầu những năm 2000, ba lập trình viên trẻ là David Helgason (CEO), Nicholas Francis (CCO), và Joachim Ante (CTO) với nguồn kinh tế eo hẹp đã tập trung tại một tầng hầm và bắt đầu lập trình ra thứ mà sau này trở thành một trong những phần mềm được ứng dụng rộng rãi nhất trong ngành công nghiệp video game.

Năm 2008, với sự gia tăng người dùng của iPhone, Unity là một trong những nhà phát triển Engine Game đầu tiên để bắt đầu hỗ trợ các nền tảng đầy đủ. Unity bây giờ hỗ trợ 24 nền tảng, bao gồm cả Oculus Rift, PlayStation 4 và Linux.

Năm 2010, IBM bắt đầu tìm hiểu Unity 3D dựa trên các plug-in trên trình duyệt web, như là một cách để truy cập vào thế giới ảo 3D từ bên trong một trình duyệt web.



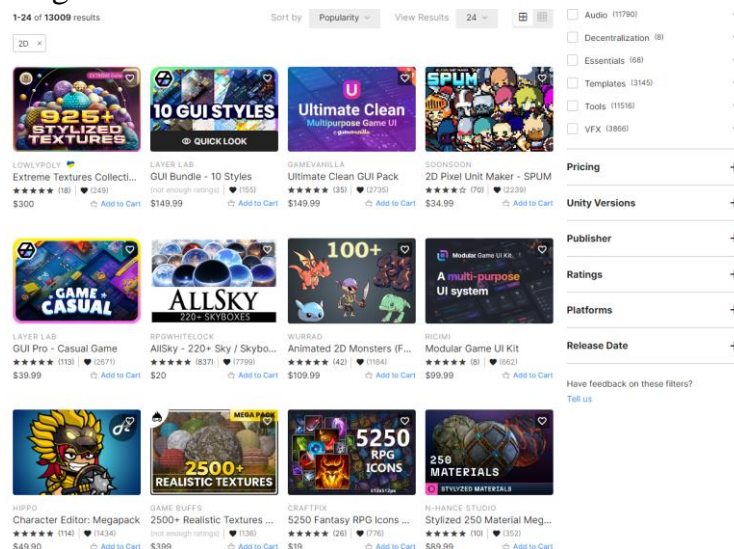
Hình 2.1. Giao diện hiện đại của Unity ngày nay

2.2. Tổng quan về các thành phần trong Unity

2.2.1. Assets

Assets là tài nguyên xây dựng nên một dự án trên Unity. Những tài nguyên có thể là hình ảnh, âm thanh, mô hình 2D 3D, chất liệu (material), texture vv hoặc cả một project hoàn chỉnh.

Các assets cũng do chính những nhà phát triển game tạo ra và có thể được download miễn phí hoặc trả phí trên Unity Asset Store. Đây là một trong những tính năng rất hay của Unity. Các asset này sẽ giúp giảm thiểu rất nhiều thời gian cho việc thiết kế và lập trình game.



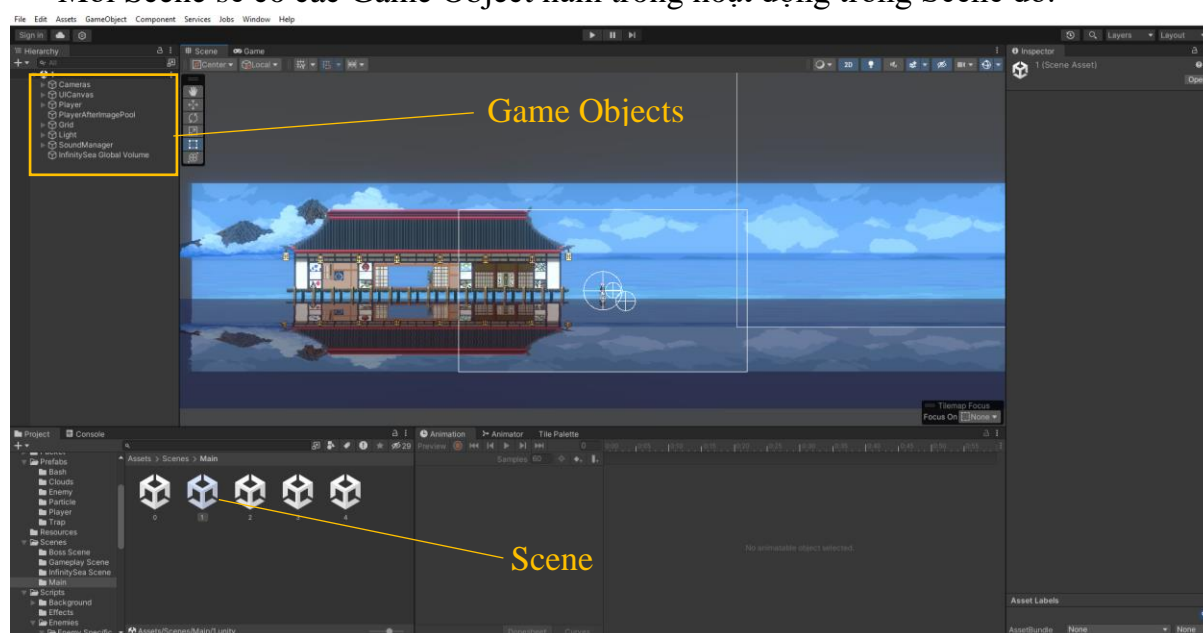
Hình 2.2. Unity Assets Store

Các assets có thể do chính chúng ta làm dự án trên Unity phát triển, tạo ra folder assets chứa những tài nguyên mà mình đã xây dựng.

2.2.2. Scenes

Trong Unity, một cảnh chơi (hoặc một phân đoạn) là những màn chơi riêng biệt, một khu vực trong game hoặc thành phần có trong nội dung của trò chơi (các menu). Các thành phần này được gọi là Scene. Bằng cách tạo ra nhiều Scenes, chúng ta có thể phân phối thời gian và tối ưu tài nguyên, kiểm tra các phân đoạn trong game một cách độc lập.

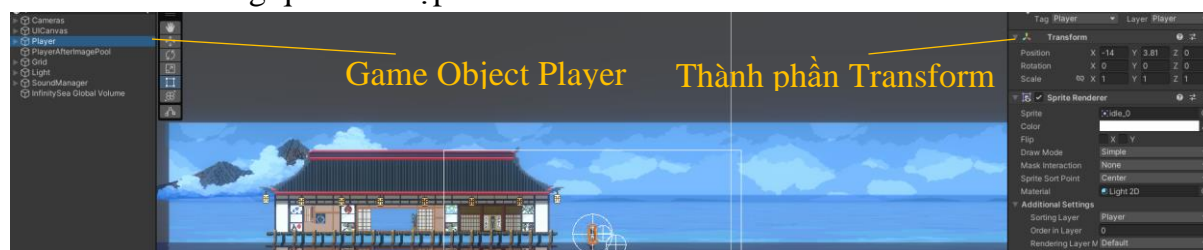
Mỗi Scene sẽ có các Game Object nằm trong hoạt động trong Scene đó.



Hình 2.3. Một Scenes trong game

2.2.3. Game Object

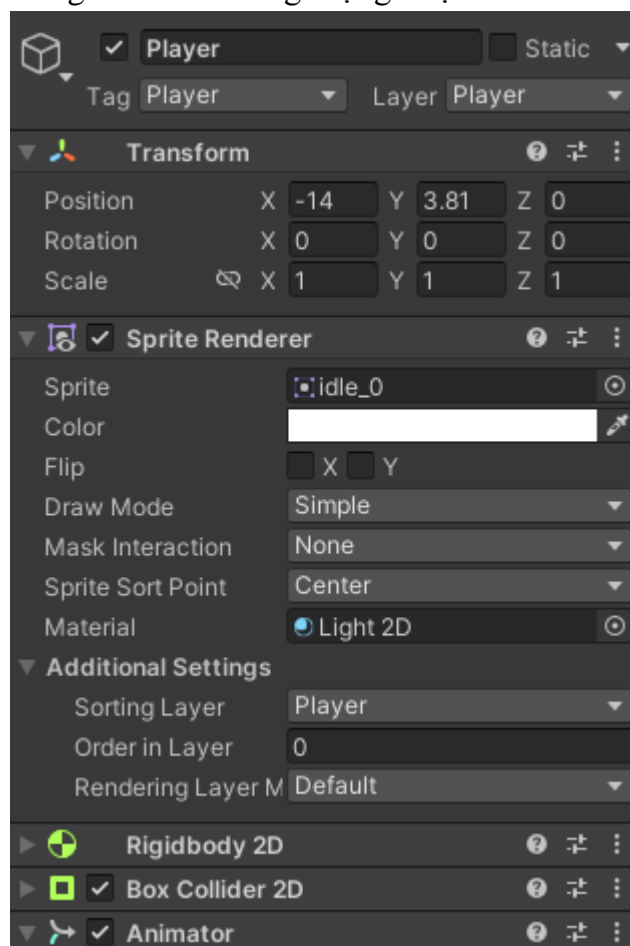
Khi Asset được sử dụng trong các Scene, Unity định nghĩa đó là Game Object. Đây là một thuật ngữ thông dụng, đặc biệt trong mảng lập trình. Tất cả các Game Object đều chứa ít nhất một thành phần cơ bản là Transform, lưu trữ thông tin về vị trí, góc xoay và tỉ lệ của Game Object. Thành phần Transform có thể được tùy biến và chỉnh sửa trong quá trình lập trình



Hình 2.4. Một Game Object

2.2.4. Components

Components là các thành phần trong game, bổ sung tính năng cho các Game Object. Mỗi Component có chức năng riêng biệt. Đa phần các Component phụ thuộc vào Transform, vì nó lưu trữ các thông số cơ bản của Game Object. Bản chất của Game Object là không có gì cả, các đặc tính và khả năng của Game Object nằm hoàn toàn trong các Component. Do đó chúng ta có thể xây dựng nên bất kỳ Game Object nào trong game mà chúng ta có thể tưởng tượng được.



Hình 2.5. Cửa sổ Components

2.2.5. Scripts

Scripts được Unity xem như một Component. Đây là thành phần thiết yếu trong quá trình phát triển game. Bất kỳ một game nào, dù đơn giản nhất đều cần đến Scripts để tương tác với các thao tác của người chơi, hoặc quản lý các sự kiện để thay đổi chiều hướng của game tương ứng với kịch bản game.

Unity cung cấp cho lập trình viên khả năng viết Script bằng các ngôn ngữ: JavaScript, C#. Unity không đòi hỏi lập trình viên phải học cách lập trình trong Unity, nhưng trong nhiều tình huống, chúng ta cần sử dụng Script trong mỗi phần của kịch bản game.

Để viết Script, chúng ta có thể làm việc với một trình biên tập Script độc lập của Unity, hoặc làm việc trên Mono Developer được tích hợp vào Unity trong những phiên bản gần đây. Mono Developer là một IDE khá tốt, cung cấp nhiều chức năng tương tự Visual Studio chúng ta cũng có thể dùng Visual Studio để viết file C# như bình thường. Mã nguồn viết trên Mono Developer sẽ được cập nhật và lưu trữ trong dự án trên Unity.



Hình 2.6. Các Scripts

2.2.6. Collider

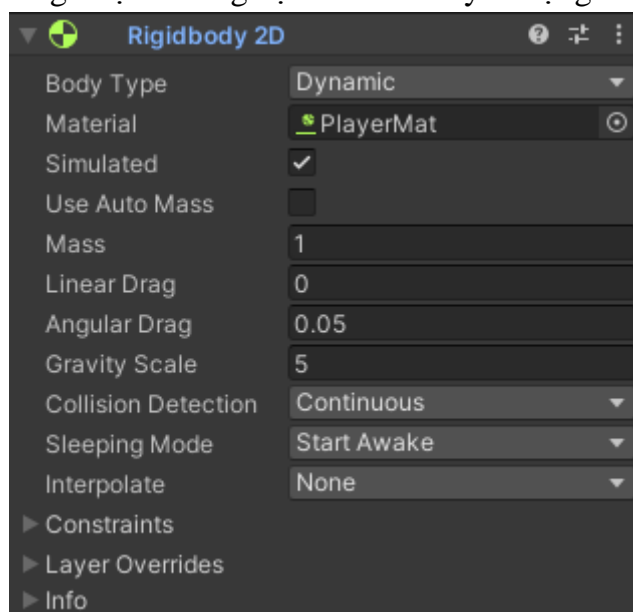
Hệ thống xử lý va chạm bao gồm 2d và 3d. Được chia ra các va chạm của các hình cơ bản:

- Box collider (phổ biến): Va chạm cho các vật hình hộp chữ nhật đối với 3d và hình chữ nhật đối với 2d.
 - Is Trigger: dạng true/false, cho phép va chạm có đi xuyên qua không?
 - Material: tham chiếu đến physic material.
 - Center: điều chỉnh thông số tâm của va chạm theo các trục tương ứng.
 - Size: điều chỉnh kích thước theo các trục tương ứng.
- Capsule Collider(phổ biến): Va chạm dành cho các khối hình trụ.
 - Is Trigger: dạng true/false, cho phép va chạm có đi xuyên qua không?
 - Material: tham chiếu đến physic material.
 - Center: điều chỉnh thông số tâm của va chạm theo các trục tương ứng.
 - Radius: điều chỉnh kích thước bán kính.
 - Height: độ cao.
 - Direction: xoay hình trụ theo các trục tương ứng.
- Box collider: Va chạm cho các vật hình hộp chữ nhật đối với 3d và hình chữ nhật đối với 2d.
- Mesh Collider: Va chạm dạng lưới dành các vật thể không xác định hình thể.
- Wheel collider: dành cho vật thể hình bánh xe. Nó sẽ mô phỏng hệ thống va chạm giống với các bánh xe.
- Terrain Collider: dành cho các vật thể địa hình và terrain collider sẽ dựa theo địa hình đó.

2.2.7. Rigidbody

Hệ thống mô phỏng vật lý trong game. Bao gồm những thông số cơ bản như:

- Body Type: Trạng thái của mô phỏng đó. Dynamic sẽ kích hoạt thúc đẩy bởi động cơ vật lý, Kinematic thì không.
- Material: Vật liệu để cấu thành mô phỏng đó.
- Simulated: dạng true/false, cho phép thực hiện hoặc không.
- Mass: khối lượng (đơn vị tùy ý).
- Linear Drag: Sức cản không khí ảnh hưởng như thế nào với đối tượng khi di chuyển. 0 có nghĩa là không có sức cản không khí, và vô cùng làm cho các đối tượng di chuyển ngay lập tức dừng lại.
- Angular Drag: Sức cản không khí ảnh hưởng đến các đối tượng khi quay từ mô-men xoắn. 0 có nghĩa là không có sức cản không khí. Không thể làm cho vật dừng quay hẳn chỉ bằng cách thiết lập Angular Drag của nó đến vô cùng.
- Gravity Scale: Độ lớn trọng lực.
- Collision detection: Được sử dụng để ngăn chặn các đối tượng chuyển động nhanh qua các đối tượng khác mà không phát hiện va chạm.
- Constraints: Ràng buộc Những hạn chế về chuyển động của Rigidbody.

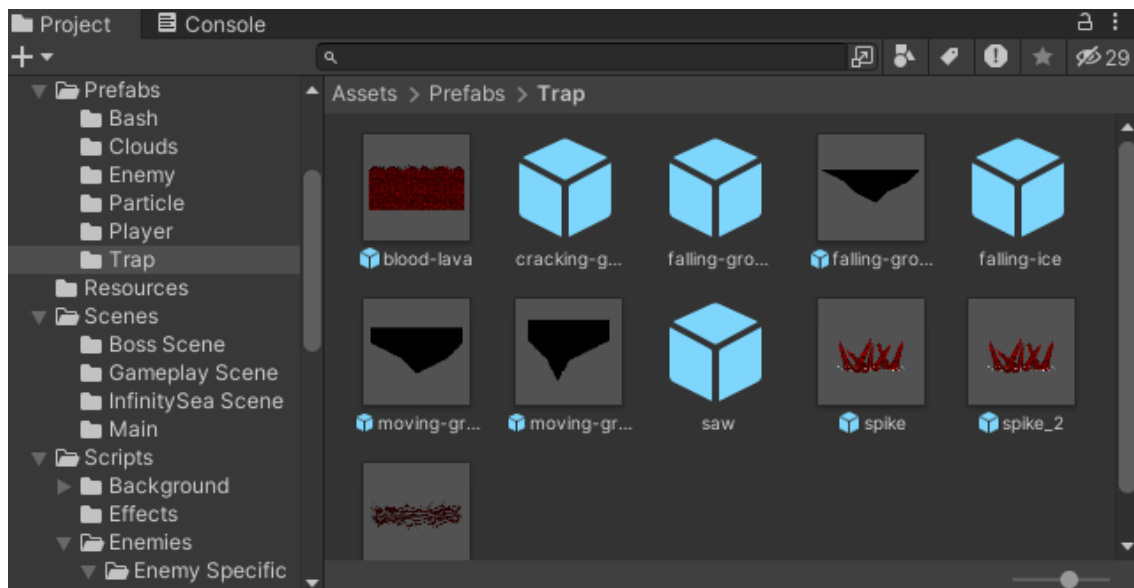


Hình 2.7. Component Rigidbody 2D

2.2.8. Prefabs

Prefabs thực chất là Game Object được lưu trữ lại để tái sử dụng. Các Game Object được nhân bản từ một prefab sẽ giống nhau hoàn toàn, ngoại trừ thành phần Transform để phân biệt và quản lý được tốt hơn.

Để tạo ra một prefab, ta đơn giản chỉ cần kéo một Game Object vào cửa sổ Project.



Hình 2.8. Cửa sổ Prefabs

2.2.9. Sprite

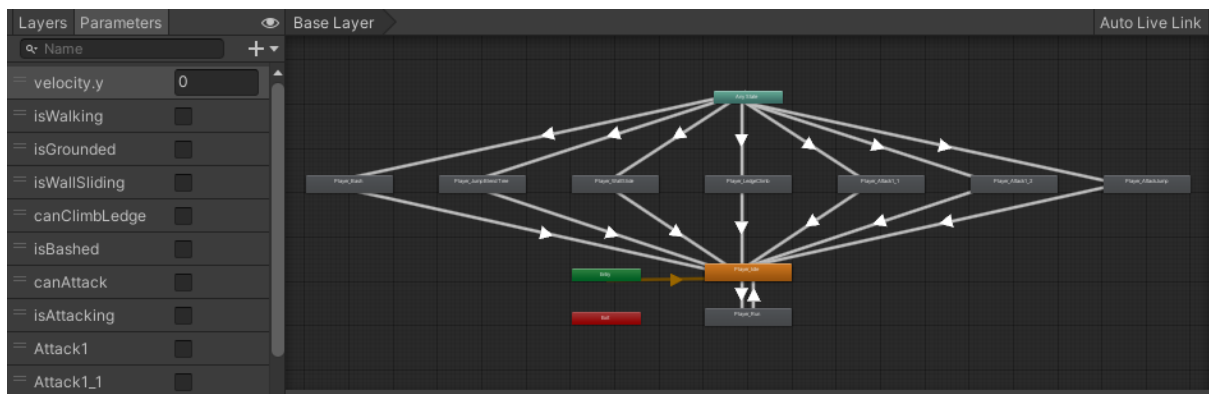
Là một hình ảnh 2D của một Game Object có thể là hình ảnh đầy đủ, hoặc có thể là một bộ phận nào đó. Unity cho phép tùy chỉnh màu sắc, kích thước, độ phân giải của một hình ảnh 2D.

2.2.10. Animator

Trong 2D thì Animation là tập một hình ảnh động dựa trên sự thay đổi liên tục của nhiều sprite khác nhau. Trong 3d là một tập hợp các sự thay đổi theo thời gian của đối tượng trong không gian. Mỗi thay đổi là một Key Frame. Key Frame hay Frame là một trạng thái của một Animation.

Để điều khiển 1 chuỗi các Animation của 1 chức năng tổng thể lớn hơn ta dùng 1 Animator gồm các thành phần:

- Controller: Bộ điều khiển animation gắn liền với nhân vật này. Nó sẽ quản lý các animation clip, các thông số tốc độ của animation, thứ tự các clip...
- Avatar: thành phần tạo hình ảnh cho object.
- Apply Root Motion: dạng true-false, cho phép thiết lập animation có di chuyển theo không gian đã được tạo khi cấu hình animation.
- Animate Physics: dạng true-false, khi được chọn, các hình ảnh trong animation sẽ có thể tương tác vật lý với nhau.
- Culling mode: chọn chế độ cho hình ảnh động.
- Parameters: các điều kiện để kích hoạt từ Animation này sang khác.

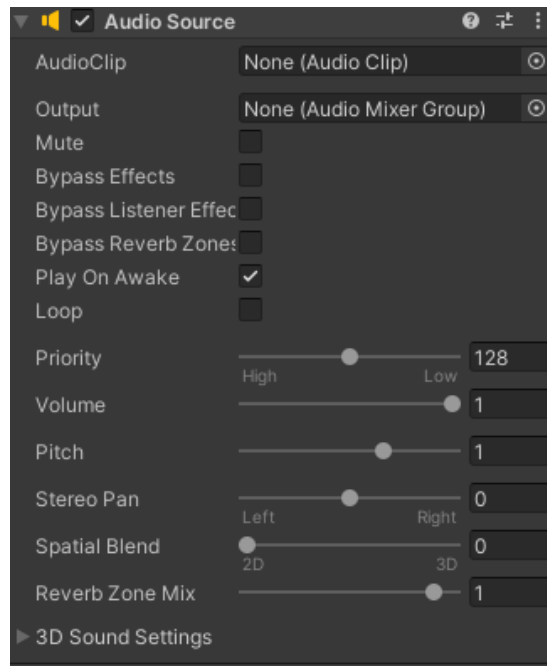


Hình 2.9. Một sơ đồ Animator

2.2.11. Audio Source

Hệ thống âm thanh trong game. Gồm cả âm thanh 2d và 3d với 1 số thành phần cơ bản sau:

- Audio clip: tham chiếu đến file âm thanh.
- Mute: chơi ở chế độ như tắt tiếng.
- Bypass effects: bộ lọc hiệu ứng áp dụng cho các nguồn âm thanh.
- Bypass listener effects: Điều này là để nhanh chóng chuyển tất cả các hiệu ứng Listener on/off.
- Pass Reverb Zones: Điều này là để nhanh chóng chuyển tất cả các khu Reverb on/off.
- Play on awake: Nếu được kích hoạt, âm thanh sẽ bắt đầu chơi lúc cảnh ra mắt. Nếu vô hiệu hóa, cần phải bắt đầu nó bằng cách sử dụng lệnh Play() từ kịch bản script.
- Loop: Kích hoạt tính năng này để làm cho Clip âm thanh lặp lại.
- Pitch: Xác định ưu tiên của nguồn âm thanh này trong số tất cả những nguồn âm cùng tồn tại trong bối cảnh đó. (Ưu tiên: 0 = quan trọng nhất, 256 = ít quan trọng nhất Mặc định = 128).
- 3D Sound Setting: Cài đặt được áp dụng cho các nguồn âm thanh nếu Audio Clip là một âm thanh 3D.
- 2D Sound Setting: Cài đặt được áp dụng cho các nguồn âm thanh nếu Audio Clip là một âm thanh 2D.

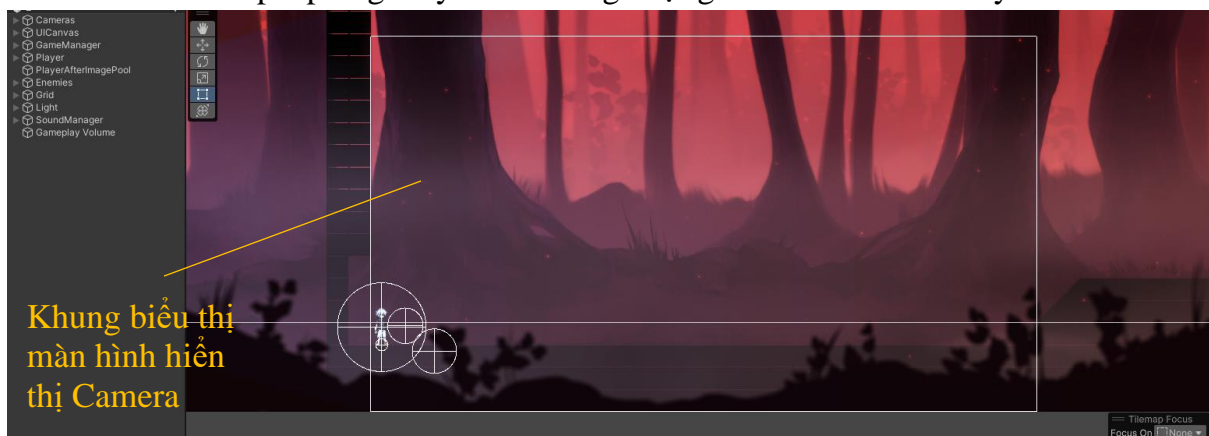


Hình 2.10. Component Audio Source

2.2.12. Camera

Là một game object đặc biệt trong scene, dùng để xác định tầm nhìn, quan sát các đối tượng khác trong game. Bao gồm các thuộc tính:

- Clear flags: Xác định phần nào của màn hình sẽ bị xóa. Đây là tiện dụng khi sử dụng nhiều máy ảnh để vẽ các yếu tố trò chơi khác nhau.
- Background: Màu áp dụng cho các màn hình còn lại sau khi tắt cả các yếu tố trong quan điểm đã được rút ra và không có skybox.
- Culling Mask: Bao gồm hoặc bỏ qua lớp của các đối tượng được đưa ra bởi các Camera.
- Projection: khả năng của máy ảnh để mô phỏng góc nhìn.
- Field of view (thuộc tính chỉ xuất hiện khi chọn Perspective trong mục Projection): Chiều rộng của góc nhìn của Camera, đo bằng độ dọc theo trục Y.
- HDR: Cho phép High Dynamic Range dựng hình cho camera này.

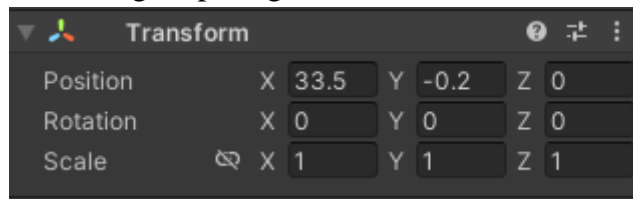


Hình 2.11. Khung biểu thị màn hình hiển thị Camera

2.2.13. Transform

Transform: quản lý object trong không gian ba chiều, theo ba thông số:

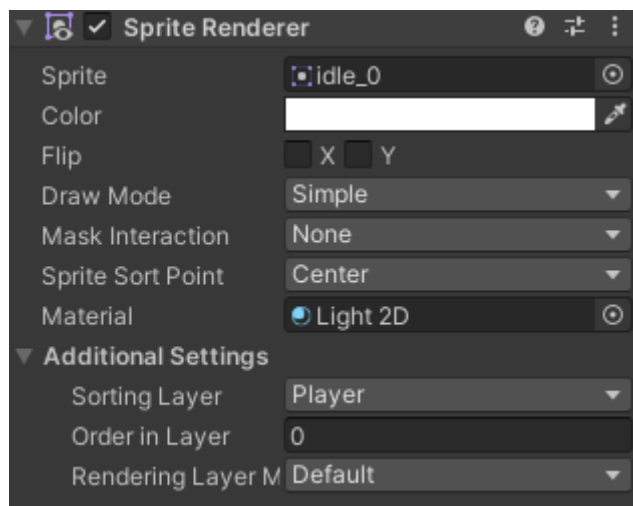
- Position: quản lý vị trí hiện tại của object.
- Rotation: quản lý các thông số quay của object theo các trục x, y, z.
- Scale: quản lý các thông số phóng to, thu nhỏ theo các trục x, y, z.



Hình 2.12. Component Transform

2.2.14. Renderer

Các SpriteRenderer thành phần cho phép bạn hiển thị hình ảnh như Sprites để sử dụng trong cả hai cảnh 2D và 3D.



Hình 2.12. Component Sprite Renderer

- Sprite: Các đối tượng Sprite để render. Đối tượng Sprite có thể được tạo ra từ textures bằng cách sử dụng các thiết lập Sprite.
- Color: Vertex màu.
- Material: Chất liệu được sử dụng để làm sprite.
- Sorting Layer: Các layer được sử dụng để xác định các ưu tiên của sprite này trong khi hiển thị.
- Order in Layer: Các ưu tiên của sprite trong layer của nó. Con số thấp hơn được kết xuất đầu tiên và con số tiếp theo phủ bên dưới.

2.3. Nguyên tắc thiết kế

2.3.1. Nguyên tắc 1: Thiết kế giao diện game chặt chẽ và dễ sử dụng

Cách thiết kế giao diện cho ứng dụng là một trong những vấn đề rất quan trọng đối với người chơi. Ngoài ra, bạn còn cần phải quan tâm đến bố cục của game sao cho hợp lý để người dùng dễ dàng tiếp cận các thông tin cần thiết khi sử dụng. Bởi ứng dụng thiết kế không tốt thì người dùng sẽ khó tiếp cận được những nội dung theo đúng ý muốn của người chơi. Từ đó, người chơi sẽ nhanh chóng thoát ra khỏi ứng dụng và cũng sẽ không muốn quay lại khi có nhu cầu.

Về cách phân chia và tổ chức, trước khi Game được thiết kế, nhóm đề tài đã định hình một số khung giao diện thường gặp và thiết kế chúng trở thành giao diện.

2.3.2. Nguyên tắc 2: Game phải được sử dụng một cách mượt mà

Các cử chỉ, hành động của nhân vật chính cũng như kẻ thù phải được thực thi một cách trơn tru, do đó tác giả đã tận dụng để xây dựng cách điều khiển bằng nút bấm trên bàn phím nhằm giúp cho người dùng có thể thao tác dễ dàng nhất.

2.3.3. Nguyên tắc 3: Cách cài đặt game phải dễ dàng

Nhờ công cụ Unity, việc cài đặt và chơi trên Windows hết sức dễ dàng, chỉ cần chọn nền mà mình mong muốn game được chạy trên đó và build.

Hệ thống sẽ tự động build cho chúng ta 1 file .exe để chúng ta có thể click double vào và game sẽ hoạt động

Chương 3. Nội dung thực hiện

3.1. Phân tích đề tài

3.1.1. Tóm tắt

- Mô tả: Tên game: **Kaneki Kiên**

Một ngạ quỷ không biết thực hư, lang thang trên hành trình vô định không có hồi kết. Cần nhiều hơn là sự thông minh, lòng dũng cảm trên con đường phát triển tìm về bản ngã và trở về lại dáng vẻ của một con người.

'Kaneki Kiên' là một tựa game action-platformer với phong cách hand-painted artwork cùng lối chơi độc đáo, đổi theo cuộc phiêu lưu trong thế giới Limbo với đầy rẫy những mối nguy hiểm nhưng cũng rất tuyệt vời này.

- Mục tiêu chính

- Tạo Ra Một Trải Nghiệm Nghệ Thuật Đẹp:

Phát triển một thế giới đẹp mắt và đầy cảm xúc, sử dụng nền đồ họa hand-painted và âm nhạc tuyệt vời để làm tăng cường trải nghiệm người chơi.

- Xây Dựng Một Câu Chuyện Sâu Sắc:

Kể một câu chuyện đầy cảm xúc về sự hy sinh, bản ngã của nhân vật, thúc đẩy người chơi suy nghĩ và cảm nhận về những giá trị nhân văn về xã hội và con người hiện nay.

- Gameplay Đa Dạng và Thách Thức:

Tạo ra một hệ thống gameplay linh hoạt, cơ chế sáng tạo, kết hợp giữa yếu tố thách thức và yếu tố phiêu lưu, khuyến khích sự sáng tạo và linh hoạt trong cách người chơi giải quyết các thử thách.

- Cảm Xúc Người Chơi:

Tạo ra sự kích thích về lối chơi khiến người chơi hòa mình vào thế giới của nhân vật, đồng thời tạo nên những mắt xích xúc cảm mạnh mẽ với người chơi.

- Đánh giá:

Mục tiêu đạt được sự đánh giá tích cực từ cả giới chuyên môn và cộng đồng người chơi.

Tăng cường uy tín và giữ cho 'Kaneki Kiên' là một trong những tựa game indie nổi bật.

- Tạo Cơ Hội Cho Các Tựa Game Tiếp Theo:

Nếu dự án thành công sẽ là bước tiến lớn cho những dự án tiếp theo dựa trên thương hiệu của tựa game.

3.1.2. Phân tích thị trường

- Xu hướng thị trường

- *Abstract*

Báo cáo nghiên cứu thị trường này tập trung vào động lực của ngành công nghiệp game, khám phá cảnh quan chung và thị trường đặc biệt của các trò chơi indie. Nghiên cứu nhằm cung cấp cái nhìn sâu sắc về xu hướng thị trường, sở thích của người chơi và cơ hội tăng trưởng tiềm năng. Qua một phân tích toàn diện, báo cáo này làm sáng tỏ về tình trạng hiện tại của ngành công nghiệp game và những thách thức và triển vọng đặc biệt liên quan đến thể loại game indie.

- **Introduction**

Ngành công nghiệp game đã trở thành một lực lượng tăng trưởng mạnh mẽ trên toàn cầu, với hàng tỷ người chơi tham gia mỗi ngày. Trong bối cảnh này, nghiên cứu thị trường là một công cụ quan trọng để hiểu rõ hơn về xu hướng tiêu dùng, sự đổi mới và cơ hội thị trường. Bài báo cáo này sẽ tập trung vào cả thị trường game nói chung và thị trường game indie nói riêng, nhấn mạnh vào những yếu tố quyết định sự thành công trong cả hai lĩnh vực.

- **Development trends Indie**

- **Ưu Điểm và Thách Thức:** Game indie nổi bật với sự độc lập và sáng tạo. Điều này mang lại ưu điểm về sự linh hoạt và khả năng thử nghiệm ý tưởng mới. Tuy nhiên, thách thức lớn nhất đối với indie là nguồn lực hạn chế và khả năng tiếp cận thị trường, khiến cho việc quảng cáo và phát triển trở nên khó khăn.

- **Đối Tượng Cạnh Tranh Trong Dòng Game Indie:**

- **Sự Đa Dạng về Nội Dung và Phong Cách:** Đối tượng cạnh tranh trong dòng game indie không chỉ đến từ những đối thủ trực tiếp mà còn xuất phát từ sự đa dạng về nội dung và phong cách. Người chơi ngày càng trở nên kỳ vọng với sự độc đáo trong từng trò chơi, từ câu chuyện sâu sắc đến cách thức tương tác độc đáo, đặt ra áp lực lớn cho những nhà phát triển indie phải liên tục sáng tạo và mang lại trải nghiệm mới lạ.

- **Xu Hướng Nghệ Thuật và Âm Nhạc:** Sự chú ý đặt vào nghệ thuật và âm nhạc cũng là một yếu tố quan trọng khi cạnh tranh trong dòng game indie. Những tựa game thành công không chỉ làm đẹp cho mắt với đồ họa, mà còn kết hợp âm nhạc sáng tạo, tạo ra một trải nghiệm đầy đủ và cuốn hút.

- **Sự Kết Hợp Giữa Ngôn Ngữ Mỹ Thuật và Công Nghệ:** Xu hướng kết hợp giữa ngôn ngữ mỹ thuật và công nghệ đang ngày càng trở nên quan trọng. Việc sử dụng công nghệ để nâng cao trải nghiệm chơi game, kết hợp với nghệ thuật sáng tạo, là một yếu tố quyết định để nổi bật trong sự cạnh tranh giữa các tựa game indie.

Việc đối mặt với những đối thủ đa dạng và không ngừng thay đổi là thách thức lớn đối với những nhà phát triển game indie. Tuy nhiên, đó cũng là động lực để họ duy trì tinh thần sáng tạo và đáp ứng sự đa dạng của người chơi, tạo ra những trải nghiệm giải trí độc đáo và không ngừng thu hút sự chú ý của cộng đồng game thủ.

- **Mục Tiêu Người Chơi Game Indie:**

- **Trải Nghiệm Độc Đáo và Tương Tác Cộng Đồng:** Người chơi indie thích thú với trải nghiệm độc đáo mà dòng game này mang lại. Sự tương tác cộng đồng là quan trọng, khi họ muốn cảm giác rằng họ đang chơi một tác phẩm không chỉ do nhà phát triển tạo ra mà còn do sự đóng góp của cộng đồng.
- **Khả Năng Tùy Chỉnh và Sáng Tạo Cá Nhân:** Mục tiêu của người chơi indie thường liên quan đến khả năng tùy chỉnh và sáng tạo cá nhân trong trò chơi. Họ muốn thấy rằng trò chơi phản ánh gu cá nhân của họ và mang đến trải nghiệm duy nhất mà họ không thể tìm thấy ở những tựa game lớn.

➤ **Đối tượng người chơi:**

- **Đối tượng người chơi mục tiêu:**
 - Người chơi yêu thích trò chơi phiêu lưu hành động: ‘Kaneki Kiên’ là một trò chơi phiêu lưu hành động với yếu tố platformer, hướng tới nhóm người chơi thích khám phá và vượt qua các thử thách trong một thế giới hư cấu.
 - Những nhóm người chơi với chế độ Multiplayer thỏa sức giải trí cùng bạn bè, trợ giúp nhau vượt qua những chướng ngại và cùng nhau tạo nên khoảng thời gian vui vẻ
 - Những người chơi yêu thích nghệ thuật đẹp và âm nhạc tuyệt vời: ‘Kaneki Kiên’ nổi tiếng với cảnh quan tuyệt đẹp và hình ảnh nghệ thuật đặc sắc, kết hợp với âm nhạc độc đáo, tạo ra một trải nghiệm thị giác và âm thanh đặc biệt.
 - Người chơi muốn trải nghiệm câu chuyện sâu sắc và cảm động: ‘Kaneki Kiên’ mang đến một câu chuyện đầy cảm xúc về tình yêu, tình bạn và sự hy sinh. Nhóm người chơi này muốn trải nghiệm một cốt truyện tốt được kể qua hình ảnh và sự tương tác.
- **Nhu cầu và mong muốn của đối tượng người chơi mục tiêu:**
 - Trải nghiệm gameplay thú vị: Đối tượng người chơi mong muốn một hệ thống gameplay đa dạng, từ việc khám phá thế giới mở đến vượt qua các thử thách và giải đố.
 - Thách thức và sự tiến bộ: Người chơi muốn gặp phải những thử thách đòi hỏi kỹ năng và sự tinh thông trong quản lý nhân vật. Đồng thời, họ mong muốn có sự tiến bộ và trở nên mạnh mẽ hơn khi vượt qua những trở ngại trong game. Ngoài ra, người chơi còn học hỏi được nhiều kiến thức bổ ích, thú vị trong quá trình khám phá game với những nguyên tố hóa học đầy kì bí.
 - Tương tác với một thế giới hư cấu đẹp: Đối tượng người chơi mong muốn khám phá và tương tác với một thế giới game đầy màu sắc và đẹp đẽ, với các môi trường độc đáo và đầy phong cách.
 - Trải nghiệm cảm xúc và lắng đọng: ‘Kaneki Kiên’ hướng đến việc tạo ra những trải nghiệm cảm xúc sâu lắng và cảm động. Đối tượng người chơi

mong muốn được truyền cảm hứng và có một trải nghiệm tâm lý sâu sắc qua câu chuyện và hình ảnh trong game.

3.1.3. Cơ chế gameplay

Platformer: là một trò chơi platformer, trong đó người chơi điều khiển nhân vật chính để vượt qua các môi trường đa dạng và thách thức.

Game platformer thường có các yếu tố cơ bản sau:

- Nhân vật chính: Thường là một nhân vật nhỏ nhẹ để nhảy nhót thoăn thoắt. Cần phải có khả năng nhảy, chạy, đi bộ.
- Môi trường: Thường là các màn chơi 2D gồm nhiều tầng lầu với nền, bậc, thanh chạy, ngăn cách bằng khoảng trống.
- Kẻ thù: Có các loại kẻ thù cần tránh hoặc đánh bại.
- Vật phẩm: Có thể cứu sống nhân vật, tăng điểm số, sức mạnh như viên nổ, sao.
- Các chế độ chơi: Vượt qua các màn để hoàn thành trò chơi hoặc chơi lại nhanh hơn, khó hơn.
- Điểm số, mạng sống: Đánh giá năng lực của người chơi dựa trên điểm số và số mạng còn lại hoặc return save.
- Âm nhạc, hình ảnh: Tạo cảm giác hồi hộp, thú vị cho người chơi.

Cấu trúc Metroidvania của trò chơi tạo ra một thế giới mở với các khu vực khám phá, yêu cầu người chơi có khả năng quay lại các địa điểm trước để mở khóa các kỹ năng mới và vượt qua các vùng đất mới.

Cấu trúc chủ đạo của thể loại game Metroidvania:

- Thế giới mở: Môi trường chơi rộng lớn gồm nhiều khu vực khác nhau. Người chơi có thể quay lại các khu vực đã khám phá.
- Khả năng nâng cấp: Nhân vật được nâng cấp theo tiến trình trò chơi nhờ vật phẩm, điểm kinh nghiệm. Mở khóa khả năng mới.
- Khu vực bị chặn: Ban đầu người chơi không thể vào được một số khu vực do thiếu khả năng, cần quay lại sau khi nâng cấp.
- Boss cuối cùng: Đánh bại boss mở khóa kết thúc trò chơi hoặc khu vực mới.
- Phát hiện bí mật: Khám phá các đường mòn, vật phẩm ngụy trang dưới lớp bề mặt.
- Câu chuyện nhỏ: Mô tả thế giới qua hồ sơ, tin nhắn, diễn biến nhỏ.

3.2. Xác định yêu cầu

3.2.1. Giao tiếp hệ thống

- Hệ thống cần phải có đầy đủ chức năng của một ứng dụng chơi game.
- Các nút điều khiển phải được liên kết chặt chẽ.

3.2.2. Giao tiếp điều khiển

- Các điều khiển phải đầy đủ, dễ dàng sử dụng để người dùng không bị ngượng khi sử dụng ứng dụng.
- Các điều khiển cần phải có hệ thống quản lý rõ ràng, liên kết chặt chẽ với nhau và đặc biệt phải phục vụ được đầy đủ các chức năng khi người dùng chơi game.

3.2.1. Giao tiếp giao diện

- Giao diện phải sử dụng các hình ảnh sắc nét, mượt mà, mới lạ, dễ nhìn.
- Giao diện phải sắp xếp các nút điều khiển, các hình ảnh, các chữ một cách ngăn nắp, gọn gàng và đặc biệt là phải phân bố hợp lý không bị loạn.

3.3. Kịch bản game

Sau khi bị nhốt và tra tấn thể xác, Kiên đã lạc lối trong chính tiềm thức của mình. Đó là một không gian vô định, đẹp đẽ và cũng đầy rẫy những cam bẫy. Hắn gặp Rize, người đã đẩy hắn ra nông nỗi này và những cảm xúc trong hắn bắt đầu đấu tranh. Nó đấu tranh dành lại phần con người đang dần mất trong hắn.

Hắn đi một quãng đường dài đầy những đẹp đẽ, đi qua những cánh cổng ngăn cách luồng ý thức giữa ngạ quỷ và con người trong hắn. Tiềm thức hắn dần nhuộm màu đỏ, màu đỏ của cái chết.

Hắn đã chết từ khi đi qua những cánh cổng torii đó, giờ hắn là một con quỷ. Con quỷ sẽ phải tự tìm đường giải thoát cho chính mình. Dù trắc trở nhưng hắn vẫn sẽ đến được cái mà hắn cho là đích, là sự thật. Hắn phải giết con quái vật do chính hắn tạo ra và trở lại với sự nghiệt ngã.

3.4. Xây dựng game

3.4.1. Xây dựng hệ thống Controller cho Player

- Hệ thống di chuyển trái, phải cho nhân vật với Animation ‘run’
- Khi người chơi ấn di chuyển thì player sẽ di chuyển trái phải, ấn nút ngược lại sẽ ‘flip’.


```

private void CheckInput()
{
    movementInputDirection = Input.GetAxisRaw("Horizontal");

    if (Input.GetButtonDown("Horizontal") && isTouchingWall)
    {
        if (!isGrounded && movementInputDirection != facingDirection)
        {
            canMove = false;
            canFlip = false;

            turnTimer = turnTimerSet;
        }
    }

    if (turnTimer >= 0)
    {
        turnTimer -= Time.deltaTime;

        if (turnTimer <= 0)
        {
            canMove = true;
            canFlip = true;
        }
    }
}

private void CheckMovementDirection()
{
    if (isFacingRight && movementInputDirection < 0 && !Mathf.Approximately(Time.timeScale, 0f))
    {
        Flip();
    }
    else if (!isFacingRight && movementInputDirection > 0 && !Mathf.Approximately(Time.timeScale, 0f))
    {
        Flip();
    }

    if (Mathf.Abs(rb.velocity.x) >= 2f)
    {
        isWalking = true;
    }
    else
    {
        isWalking = false;
    }
}

2 references
private void Flip()
{
    if (canFlip && !isWallSliding && !ledgeDetected && !knockback)
    {
        facingDirection *= -1;
        isFacingRight = !isFacingRight;
        transform.Rotate(0.0f, 180.0f, 0.0f);
    }
}

```

Hình 3.1. Script Horizaltol

- Hệ thống nhảy cho nhân vật với Animation ‘jump’
- Khi người chơi ấn nhảy, player sẽ thực hiện hành động ‘jump’, khi chạm môi trường Ground mới được thực hiện nhảy lần tiếp.
- Khi người chơi ấn nhảy mà player ở trên mặt đất sẽ thực hiện ‘normal jump’. Đơn giản là player sẽ nhảy lên theo hướng mặt player.

```

private void CheckJump()
{
    if(jumpTimer > 0)
    {
        //WallJump
        if(!isGrounded && isTouchingWall && movementInputDirection != 0 && movementInputDirection != facingDirection)
        {
            WallJump();
        }
        else if (isGrounded)
        {
            NormalJump();
        }
    }

    if(isAttemptingToJump)
    {
        jumpTimer -= Time.deltaTime;
    }

    if(wallJumpTimer > 0)
    {
        if(hasWallJumped && movementInputDirection == -lastWallJumpDirection)
        {
            rb.velocity = new Vector2(rb.velocity.x, 0.0f);
            hasWallJumped = false;
        }else if(wallJumpTimer <= 0)
        {
            hasWallJumped = false;
        }
        else
        {
            wallJumpTimer -= Time.deltaTime;
        }
    }
}

2 references
private void NormalJump()
{
    if (canNormalJump)
    {
        rb.velocity = new Vector2(rb.velocity.x, jumpForce);
        amountOfJumpsLeft--;
        jumpTimer = 0;
        isAttemptingToJump = false;
        checkJumpMultiplier = true;
    }
}

```

Hình 3.2. Script CheckJump và NormalJump

- Khi người chơi ấn nhảy mà player đang trượt trên tường sẽ thực hiện ‘wall jump’. Wall Jump chỉ cho phép player nhảy theo hướng ngược lại so với hướng đang trượt trên tường.

```

private void WallJump()
{
    if (canWallJump)
    {
        rb.velocity = new Vector2(rb.velocity.x, 0.0f);
        isWallSliding = false;
        amountOfJumpsLeft = amountOfJumps;
        amountOfJumpsLeft--;
        Vector2 forceToAdd = new Vector2(wallJumpForce * wallJumpDirection.x * movementInputDirection, wallJumpForce * wallJumpDirection.y);
        rb.AddForce(forceToAdd, ForceMode2D.Impulse);
        jumpTimer = 0;
        isAttemptingToJump = false;
        checkJumpMultiplier = true;
        turnTimer = 0;
        canMove = true;
        canFlip = true;
        hasWallJumped = true;
        wallJumpTimer = wallJumpTimerSet;
        lastWallJumpDirection = -facingDirection;
    }
}

```

Hình 3.3. Script WallJump

- Hệ thống trượt tường cho nhân vật với Animation ‘wall slide’
- Khi người chơi nhảy lên một bức tường, ấn giữ hướng di chuyển khi tiếp xúc thì player sẽ thực hiện ‘wall slide’

```
private void CheckIfWallSliding()
{
    if (isTouchingWall && movementInputDirection == facingDirection && rb.velocity.y < 0 && !canClimbLedge)
    {
        isWallSliding = true;
    }
    else
    {
        isWallSliding = false;
    }
}
```

Hình 3.4. Script Check WallSlide

- Hệ thống leo vách cho nhân vật với Animation ‘ledge climb’
- Khi player chạm vào các mép góc tường sẽ thực hiện hành động ‘ledge climb’

```
private void CheckLedgeClimb()
{
    if (LedgeDetected && !canClimbLedge)
    {
        canClimbLedge = true;

        if (isFacingRight)
        {
            ledgePos1 = new Vector2(Mathf.Floor(ledgePosBot.x + wallCheckDistance) - ledgeClimbXOffset1, Mathf.Floor(ledgePosBot.y) + ledgeClimbYOffset1);
            ledgePos2 = new Vector2(Mathf.Floor(ledgePosBot.x + wallCheckDistance) + ledgeClimbXOffset2, Mathf.Floor(ledgePosBot.y) + ledgeClimbYOffset2);
        }
        else
        {
            ledgePos1 = new Vector2(Mathf.Ceil(ledgePosBot.x - wallCheckDistance) + ledgeClimbXOffset1, Mathf.Floor(ledgePosBot.y) + ledgeClimbYOffset1);
            ledgePos2 = new Vector2(Mathf.Ceil(ledgePosBot.x - wallCheckDistance) - ledgeClimbXOffset2, Mathf.Floor(ledgePosBot.y) + ledgeClimbYOffset2);
        }

        canMove = false;
        canFlip = false;

        anim.SetBool("canClimbLedge", canClimbLedge);
    }

    if (canClimbLedge)
    {
        transform.position = ledgePos1;
    }
}

0 references
public void FinishLedgeClimb()
{
    canClimbLedge = false;
    transform.position = ledgePos2;
    canMove = true;
    canFlip = true;
    ledgeDetected = false;
    anim.SetBool("canClimbLedge", canClimbLedge);
}
```

Hình 3.5. Script LedgeClimb

- Hệ thống Ability cho nhân vật
- Khi người chơi ấn Dash, player sẽ thực hiện hành động ‘dash’, tăng tốc độ di chuyển lên gấp đôi và xuyên qua quái vật cũng như một số cạm bẫy.

```

private void AttemptToDash()
{
    isDashing = true;
    dashTimeLeft = dashTime;
    lastDash = Time.time;


    PlayerAfterImagePool.Instance.GetFromPool();
    lastImageXpos = transform.position.x;
}
1 reference
private void CheckDash()
{
    if (isDashing)
    {
        if (dashTimeLeft > 0)
        {
            canMove = false;
            canFlip = false;
            rb.velocity = new Vector2(dashSpeed * facingDirection, 0.0f);
            dashTimeLeft -= Time.deltaTime;

            if (Mathf.Abs(transform.position.x - lastImageXpos) > distanceBetweenImages)
            {
                PlayerAfterImagePool.Instance.GetFromPool();
                lastImageXpos = transform.position.x;
            }
        }

        if (dashTimeLeft <= 0 || isTouchingWall)
        {
            isDashing = false;
            canMove = true;
            canFlip = true;
        }
    }
}

```

Hình 3.6. Script Dash

- Khi người chơi tương tác với biểu tượng  sẽ thực hiện hành động ‘dash’, cho phép player lấy thêm một quãng đà và tiếp tục nhảy.

```

void CheckBash()
{
    RaycastHit2D[] Rays = Physics2D.CircleCastAll(transform.position, BashRadius, Vector3.forward);
    foreach (RaycastHit2D ray in Rays)
    {
        NearToBashAbleObj = false;

        if (ray.collider.tag == "BashAble")
        {
            NearToBashAbleObj = true;
            BashAbleObj = ray.collider.transform.gameObject;
            break;
        }
    }
    if (NearToBashAbleObj)
    {
        if (Input.GetKeyDown(KeyCode.Mouse1))
        {
            rb.velocity = Vector2.zero;
            //Instantiate(playerBash, rb.transform.position, rb.transform.rotation);
            Time.timeScale = 0;
            BashAbleObj.transform.localScale = new Vector2(1.2f, 1.2f);
            Arrow.SetActive(true);
            Arrow.transform.position = BashAbleObj.transform.position;
            IsChosingDir = true;
        }
        else if (IsChosingDir && Input.GetKeyUp(KeyCode.Mouse1))
        {
            Time.timeScale = 1f;
            BashAbleObj.transform.localScale = new Vector2(0.8f, 0.8f);
            IsChosingDir = false;
            IsBashing = true;
            rb.velocity = Vector2.zero;
            transform.position = BashAbleObj.transform.position;
            BashDir = Camera.main.ScreenToWorldPoint(Input.mousePosition) - transform.position;
            BashDir.z = 0;
            //if (BashDir.x > 0)
            //{
            //    transform.eulerAngles = new Vector3(0, 0, 0);
            //}
            //else
            //{
            //    transform.eulerAngles = new Vector3(0, 180, 0);
            //}
            BashDir = BashDir.normalized;
            BashAbleObj.GetComponent<Rigidbody2D>().AddForce(-BashDir * 5, ForceMode2D.Impulse);
            Arrow.SetActive(false);
        }
    }
}

}
else if (BashAbleObj != null)
{
    BashAbleObj.GetComponent<SpriteRenderer>().color = Color.white;
}

///// Preform the bash
if (IsBashing)
{
    if (BashTime > 0)
    {
        BashTime -= Time.deltaTime;
        rb.velocity = BashDir * BashPower * Time.deltaTime;
    }
    else
    {
        IsBashing = false;
        BashTime = BashTimeReset;
        rb.velocity = new Vector2(rb.velocity.x, 0);
    }
}
}

```

Hình 3.7. Script Bash

3.4.2. Xây dựng hệ thống Basic Combat cho Player

- Khi ở trạng thái được tấn công (Combat Enabled), người chơi ấn chuột trái (mouse0) để thực hiện tấn công. Trạng thái không được tấn công bao gồm: nhảy, trượt tường, leo góc và khi sử dụng các Ability.
- Khi người chơi ấn liên tiếp chuột trái (mouse0), 2 hành động tấn công khác nhau sẽ được đan xen và có độ trễ giữa các lần tấn công.

```
private void CheckCombatInput()
{
    if (Input.GetMouseButtonDown(0))
    {
        if (combatEnabled)
        {
            //Attempt combat
            gotInput = true;
            lastInputTime = Time.time;
        }
    }
}

1 reference
private void CheckAttacks()
{
    if (gotInput)
    {
        if (!isAttacking)
        {
            gotInput = false;
            isAttacking = true;
            anim.SetBool("isAttacking", isAttacking);
            isAttack1_1 = !isAttack1_1;
            anim.SetBool("Attack1", true);
            anim.SetBool("Attack1_1", isAttack1_1);
            if(isAttack1_1) SoundManager.instance.PlaySound(attack1_1Sound);
            if(!isAttack1_1) SoundManager.instance.PlaySound(attack1_2Sound);
            //if (isGrounded)
            //{
            //    isAttack1_1 = !isAttack1_1;
            //    anim.SetBool("Attack1", true);
            //    anim.SetBool("Attack1_1", isAttack1_1);
            //}
            //else
            //{
            //    anim.SetBool("AttackJump", true);
            //}
        }
    }

    if(Time.time >= lastInputTime + inputTimer)
    {
        //Wait for new input
        gotInput = false;
    }
}
```

Hình 3.8. Script CheckAttack

- Khi tấn công, player sẽ có 1 vùng hitbox. Khi enemies trong phạm vi này sẽ bị tấn công.
- Cần 1 hàm để có thể kết thúc trạng thái tấn công (không phải trạng thái có thể được tấn công).

```
private void CheckAttackHitBox()
{
    Collider2D[] detectedObjects = Physics2D.OverlapCircleAll(attackHitBoxPos.position, attack1Radius, whatIsDamageable);

    attackDetails.position = transform.position;
    attackDetails.damageAmount = attack1Damage;
    attackDetails.stunDamageAmount = stunDamageAmount;

    foreach (Collider2D collider in detectedObjects)
    {
        collider.transform.parent.SendMessage("Damage", attackDetails);
    }
}

0 references
private void FinishAttack1()
{
    isAttacking = false;
    anim.SetBool("isAttacking", isAttacking);
    anim.SetBool("Attack1", false);
}
```

Hình 3.9. Script Hitbox

- Khi bị tấn công, người chơi sẽ bị mất máu và lùi lại. 1 hàm Damage sẽ gọi đến struct AttackDetails lưu lượng damage nhận vào...

```
private void Damage(AttackDetails attackDetails)
{
    if(!PC.GetDashStatus())
    {
        int direction;

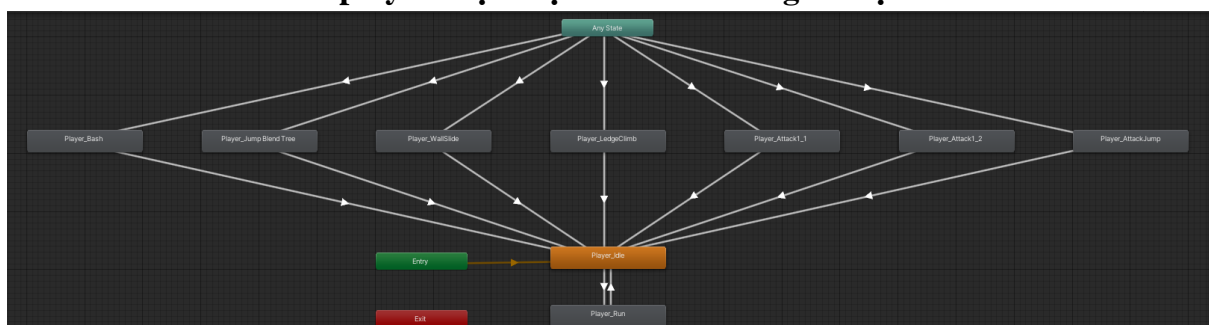
        PS.DecreaseHeath(attackDetails.damageAmount);

        if (attackDetails.position.x < transform.position.x)
        {
            direction = 1;
        }
        else
        {
            direction = -1;
        }

        PC.Knockback(direction);
    }
}
```

Hình 3.10. Script Damage

* Sơ đồ Animator cho player thực hiện các chức năng đã liệt kê ở bên trên



Hình 3.11. Sơ đồ Animator cho Player

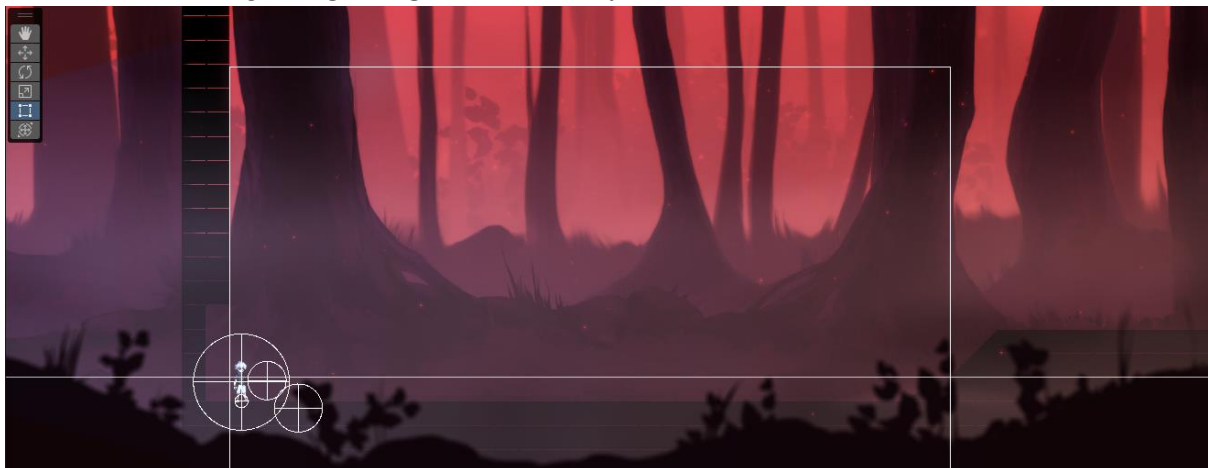
- Để thực hiện chuyển Animation, sử dụng các Parameter làm điều kiện cho các hành động.

= velocity.y	0
= isWalking	<input type="checkbox"/>
= isGrounded	<input type="checkbox"/>
= isWallSliding	<input type="checkbox"/>
= canClimbLedge	<input type="checkbox"/>
= isBashed	<input type="checkbox"/>
= canAttack	<input type="checkbox"/>
= isAttacking	<input type="checkbox"/>
= Attack1	<input type="checkbox"/>
= Attack1_1	<input type="checkbox"/>
= AttackJump	<input type="checkbox"/>

Hình 3.12. Các Parameter cho Player

3.4.3. Xây dựng camera theo dõi Player

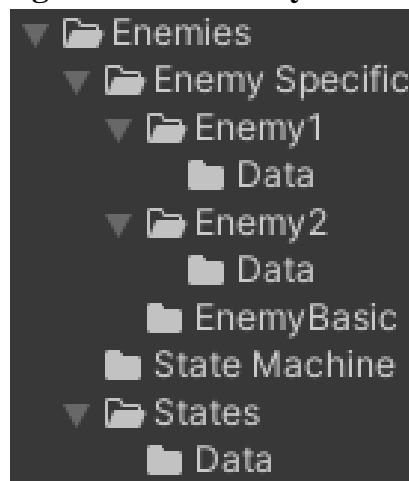
- Sử dụng Component Cinemachine trong Packet Manager của Unity, Camera sẽ tự động di chuyển khi player di chuyển. Trong dự án này, camera sẽ chiếu theo trung tâm của màn hình.
- Camera là khung trắng trong hình dưới đây



Hình 3.13. Camera trong Gameplay

- Khi gặp các góc chết (góc dừng camera) sẽ có 1 collider cambound bao lấy và không cho camera vượt quá va chạm này (sự va chạm giữa camera và collider cambound).

*** Các folder phân vùng trong thiết kế 1 Enemy**



Hình 3.14. Folder Enemies

- State Machine sẽ gồm các script Entity (thực thể), FiniteState (giới hạn các bước), State (các hàm của 1 bước)
- States sẽ gồm các script class State thực hiện các chức năng của 1 enemy bất kỳ. 1 folder Data gồm các class Data lưu các chỉ số của từng State.
- Enemy Specific gồm các class kế thừa các class State ở trên tham chiếu đến 1 loại enemy để thực hiện chức năng của loại enemy đó. 1 folder Data lưu các asset được setup từ các script Data trong State để dùng cho loại enemy đó.

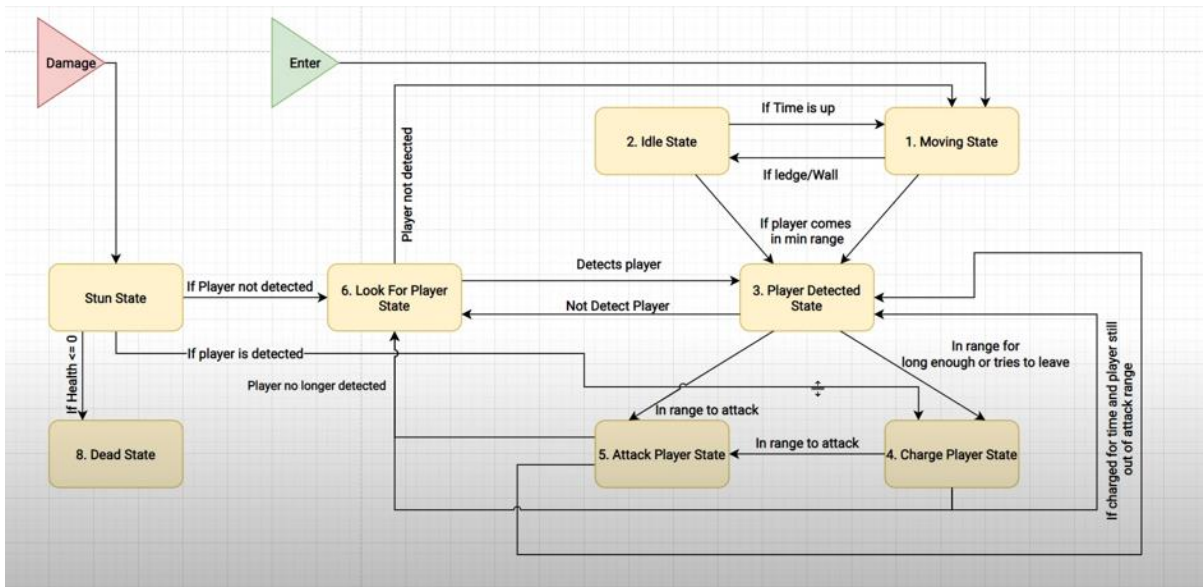
3.4.4. Xây dựng AI Enemy MeleeAttack và sơ đồ Animator

- Trong dự án gọi là Enemy1, gồm các states sau: idle (đứng yên), moving (di chuyển), player detected (phát hiện ra player), charge player (lao đến player), attack player (tấn công player), look for player (nhìn player), stun (choáng), dead (chết)



Hình 3.15. Folder Enemy Specific của Enemy1

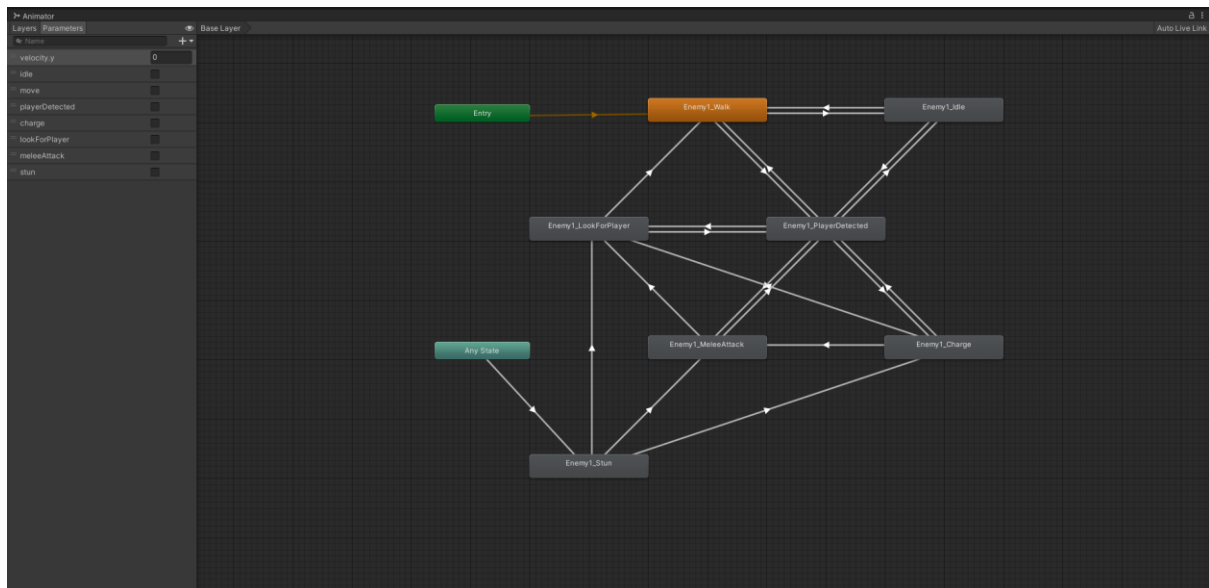
➤ Sơ đồ quan hệ giữa các lớp khi thiết kế AI Enemy MeleeAttack



Hình 3.16. Sơ đồ quan hệ giữa các lớp khi thiết kế AI Enemy MeleeAttack

- Trạng thái khởi đầu là State Moving, Enemy1 sẽ di chuyển trái-phải. Khi gặp wall (tường) hoặc ledge (vách) Enemy1 sẽ chuyển sang Idle.
- Khi đứng yên, sau 1 khoảng thời gian sẽ Flip (quay đầu) và chuyển sang Moving.
- 2 States trên khi gặp player trong 1 **phạm vi phát hiện** sẽ chuyển sang Player Detected. Lúc này, Enemy1 sẽ đứng yên 1 khoảng thời gian.
- Nếu trong lúc đó, player vẫn trong **phạm vi phát hiện**, Enemy1 sẽ chuyển sang Charge lao đến player. Nếu player ra khỏi **phạm vi lao đến** nhưng vẫn trong **phạm vi phát hiện**, Enemy1 chuyển sang Player Detected.
- 2 States trên mà player trong **phạm vi tấn công**, chuyển sang Attack. Nếu không chuyển lại về 2 States trên.
- Khi player ra ngoài **phạm vi phát hiện**, chuyển sang Look For Player. Nếu player lại trong **phạm vi phát hiện**, chuyển sang Player Detected, không thì chuyển sang Moving.
- Khi bị tấn công, tương tự như player sẽ trừ máu nhưng thanh máu không hiện mà để người chơi tự cảm nhận. Khi bị tấn công 1 số lần nhất định sẽ bị Stun.
- Khi máu về 0, chuyển sang Dead.

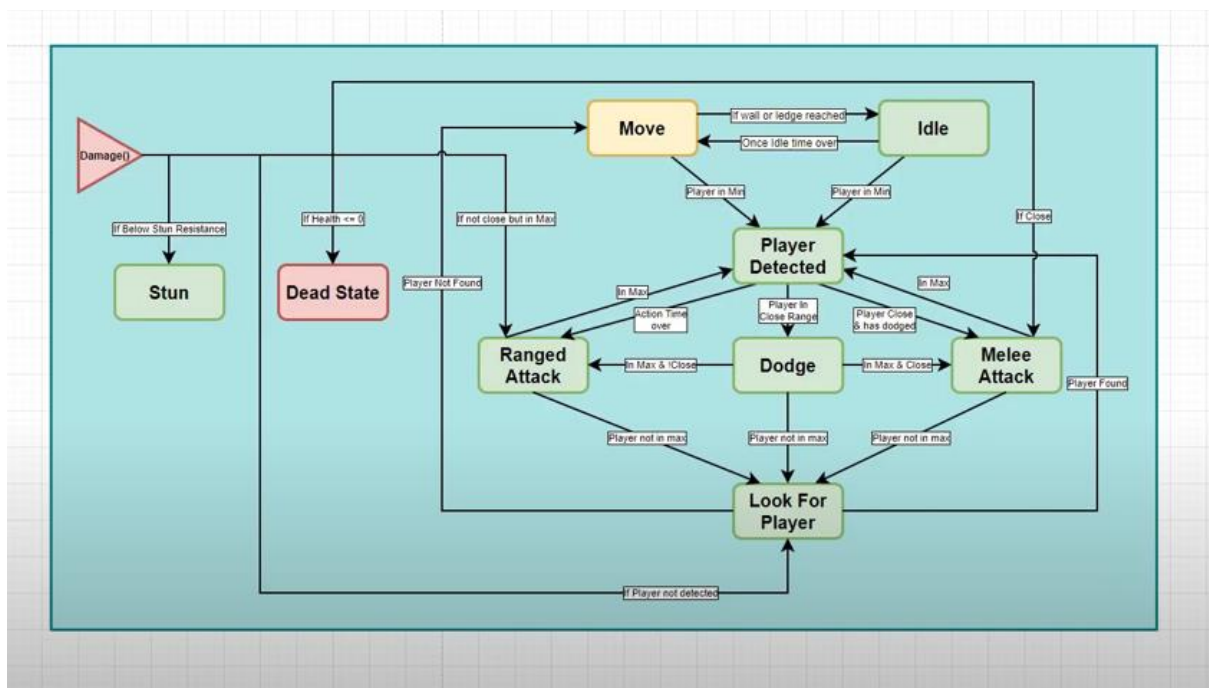
➤ Sơ đồ Animator cho Enemy1 thực hiện các chức năng đã liệt kê ở bên trên và các Parameters.



Hình 3.17. Sơ đồ Animator cho Enemy1

3.4.5. Xây dựng AI Enemy RangedAttack và sơ đồ Animator

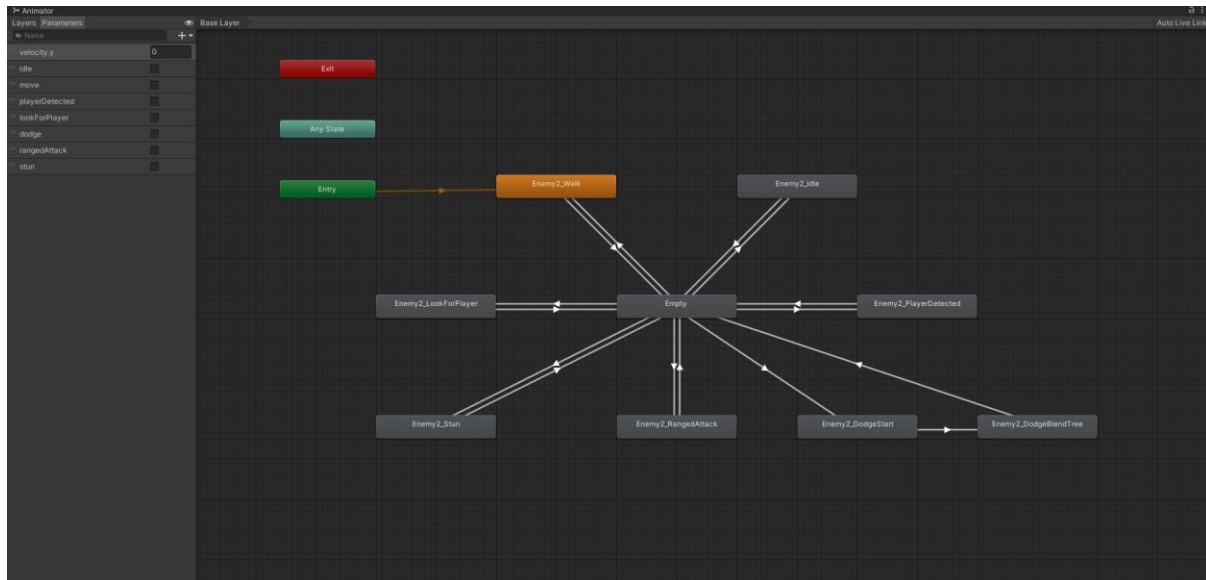
- Trong dự án gọi là Enemy2, gồm các states sau: idle (đứng yên), moving (di chuyển), player detected (phát hiện ra player), dodge (lùi lại khi gặp player), melee attack player (tấn công cận chiến), ranged attack player (tấn công cận chiến), look for player (nhìn player), stun (choáng), dead (chết)
- Sơ đồ quan hệ giữa các lớp khi thiết kế AI Enemy RangedAttack



Hình 3.18. Sơ đồ quan hệ giữa các lớp khi thiết kế AI Enemy RangedAttack

- Các States cũng gần như tương tự như bên trên
- Khi trong phạm vi gần, Enemy2 sẽ lùi về phía sau 1 chút (Dodge)
- Ở trong tầm Ranged Attack, Enemy2 sẽ gọi ra 1 Projectile là 1 viên bom từ xa. Viên bom này cũng có các Component va chạm hay vật lý.

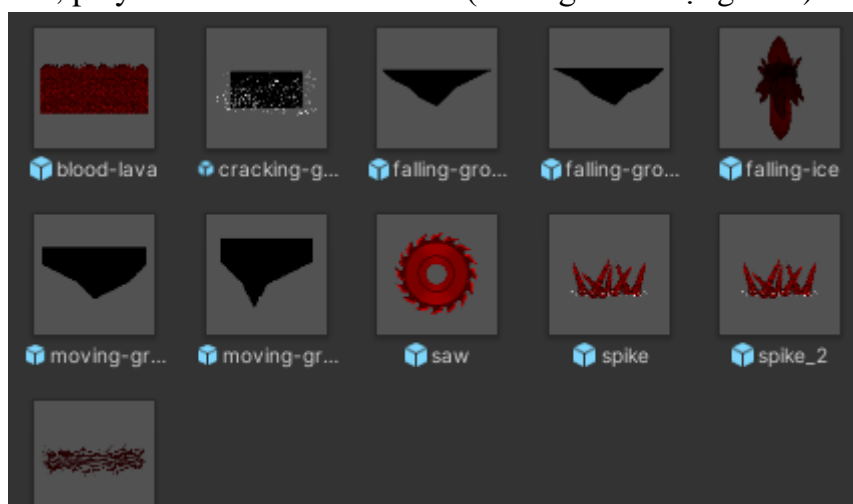
➤ Sơ đồ Animator cho Enemy1 thực hiện các chức năng đã liệt kê ở bên trên và các Parameters.



Hình 3.19. Sơ đồ Animator cho Enemy2

3.4.6. Xây dựng các chương ngại vật

- Có rất nhiều các cạm bẫy trong game mà người chơi phải vượt qua.
- Nếu chạm vào, player sẽ mất rất nhiều máu (thường là ½ lượng máu)



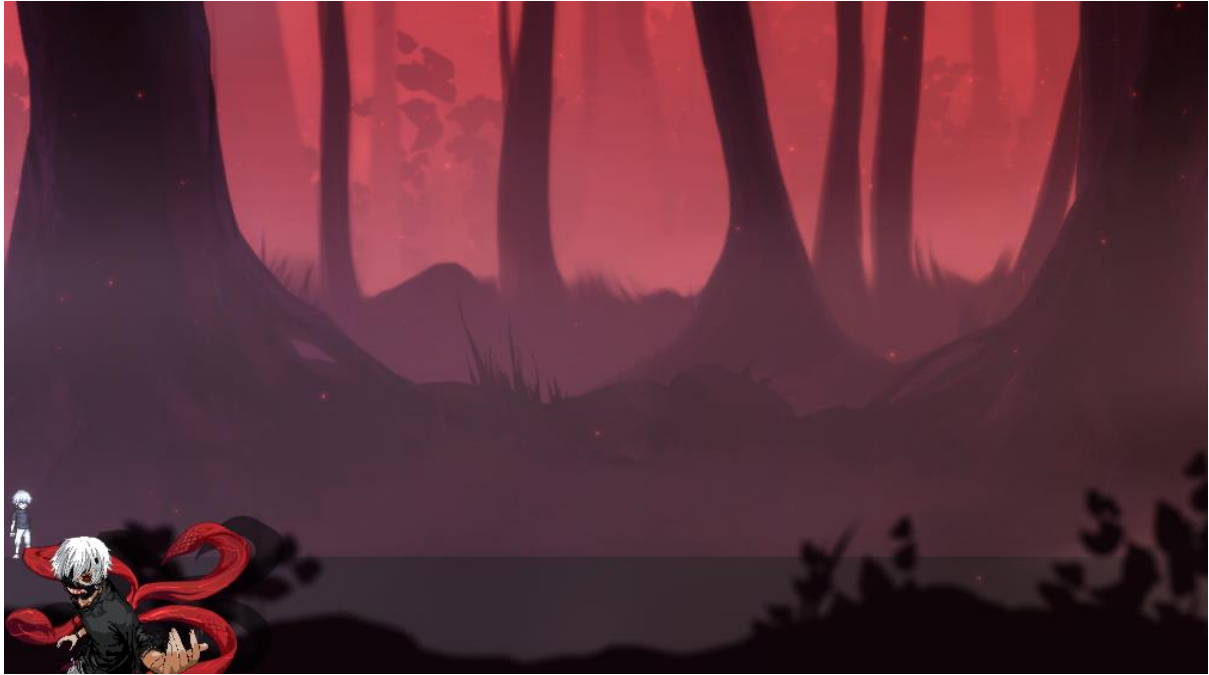
Hình 3.20. Một số Prefabs Trap

3.4.7. Xây dựng thông báo

- Là các lời thoại khi player tương tác với 1 số Game Objects, có thể là lời thoại hoặc là thông báo hướng dẫn khi gặp vật thể mới.

3.4.8. Xây dựng các phân cảnh và chuyển cảnh

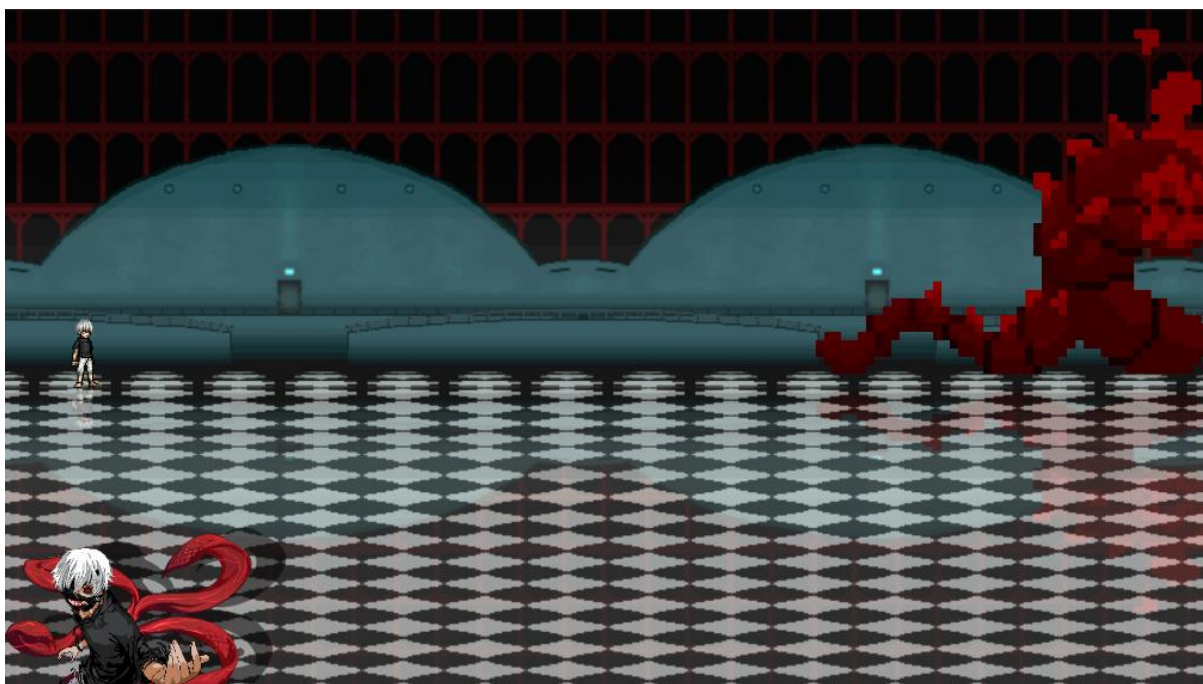
- 1 số phân cảnh có trong game:



Hình 3.21. Một số Scenes trong game



Hình 3.22. Một số Scenes trong game



Hình 3.23. Một số Scenes trong game

- Có các hiệu ứng trong từng cảnh khác nhau như hiệu ứng ánh sáng, Bloom, Shader để tạo môi trường phản chiếu...
- Khi gặp Finish Point (để trống), sẽ chuyển sang Scenes tiếp theo của gameplay. Hiệu ứng Fade (mờ dần) sẽ được sử dụng.

3.4.8. Xây dựng hệ thống âm thanh

- Có nhạc nền cho Main Menu và từng Scenes khác nhau
- Có các Sound Effect khi combat hoặc CheckPoint.
- Hệ thống âm thanh sẽ điều chỉnh âm lượng, độ lệch...

3.5. Demo Game

3.5.1. Giao diện màn hình chờ Menu



Hình 3. Màn hình Main Menu

- Khi người chơi bắt đầu vào Game hoặc là khi đang chơi out ra màn hình Menu
 - New Game: bắt đầu trò chơi mới.
 - Continue: Tiếp tục chơi tại chỗ đã Respawn.
 - Options: Điều chỉnh một số tính năng
 - Quit: Thoát game.
- Khi mà người chơi chọn mũi tên lên, xuống hay w, s thì 1 select sẽ được gọi và chỉ vào vị trí hiện tại. Nếu người chơi ấn Space hay Enter thì sẽ gọi ra chức năng mà con trỏ đang chỉ tới.

```

public class MainMenuSelect : MonoBehaviour
{
    [SerializeField] private RectTransform[] options;
    [SerializeField] private Transform[] trans;
    [SerializeField] private Shadow[] shadows;
    [SerializeField] private AudioClip changeSound;
    [SerializeField] private AudioClip interactSound;
    private RectTransform rect;
    private int currentPosition;

    [Unity Message] [0 references]
    private void Awake()
    {
        rect = GetComponent<RectTransform>();

        currentPosition = 1;
        trans[currentPosition].localScale = new Vector3(1.2f, 1.2f, 1f);
        shadows[currentPosition].effectColor = new Color(1f, 1f, 1f);
    }

    [Unity Message] [0 references]
    private void Update()
    {
        if (Input.GetKeyDown(KeyCode.W) || Input.GetKeyDown(KeyCode.UpArrow))
        {
            ChangePosition(-1);
        }
        else if (Input.GetKeyDown(KeyCode.S) || Input.GetKeyDown(KeyCode.DownArrow))
        {
            ChangePosition(1);
        }

        if (Input.GetKeyDown(KeyCode.KeypadEnter) || Input.GetKeyDown(KeyCode.Return) || Input.GetKeyDown(KeyCode.Space))
            Interact();
    }

    private void ChangePosition(int change)
    {
        currentPosition += change;

        if (change != 0)
        {
            SoundManager.instance.PlaySound(changeSound);
        }

        if (currentPosition < 0)
        {
            currentPosition = options.Length - 1;
        }
        else if (currentPosition > options.Length - 1)
        {
            currentPosition = 0;
        }

        rect.position = new Vector3(rect.position.x, options[currentPosition].position.y, 0);

        trans[currentPosition].localScale = new Vector3(1.2f, 1.2f, 1f);
        for (int i = 0; i < trans.Length; i++)
        {
            if (i != currentPosition)
            {
                trans[i].localScale = new Vector3(1f, 1f, 1f);
            }
        }

        shadows[currentPosition].effectColor = new Color(1f, 1f, 1f);
        for (int i = 0; i < trans.Length; i++)
        {
            if (i != currentPosition)
            {
                shadows[i].effectColor = new Color(0f, 0f, 0f);
            }
        }
    }

    [1 reference]
    private void Interact()
    {
        SoundManager.instance.PlaySound(interactSound);

        options[currentPosition].GetComponent<Button>().onClick.Invoke();
    }
}

```

Hình 3. Script Select

3.5.2. Màn hình Pause Game

- Khi người chơi đang trong màn chơi ấn Escape sẽ gọi đến giao diện Pause Game



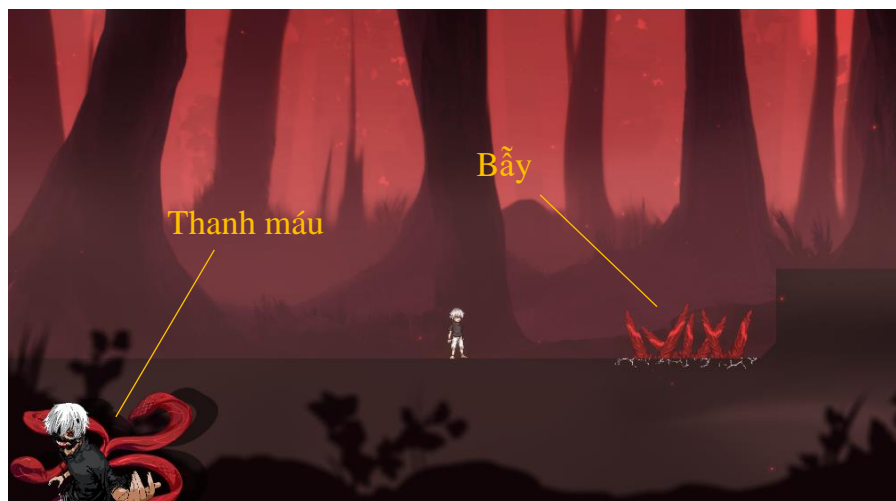
Hình 3. Màn hình Pause

- Select cũng tương tự như phần Main Menu
 - Continue: Tiếp tục chơi tại chỗ đã Pause.
 - Restart: Bắt đầu lại màn chơi.
 - Main Menu: Chuyển đến màn hình Main Menu

3.5.4. Màn hình Game Play

- Điều khiển nhân vật như đã đề cập ở bên trên bao gồm:
 - Di chuyển: trái – phải (horizontal), nhảy lên (jump)
 - Tấn công: chuột trái (mouse 0)
 - Trạng thái: thanh máu như ở trong hình
 - Chức năng: Lướt (Dash), Đập vào vật có thể tương tác (Bash)
- Map được thiết kế như sau:
 - Background: sử dụng hiệu ứng ánh sáng, hiệu ứng Bloom, hiệu ứng Parallax và một số Game Objects phụ.
 - Platforms: là layer ground, giúp player tương tác vật lý với nền.
 - Decoration: Những Game Objects hỗ trợ trước hoặc sau Platform giúp game chân thực và đẹp mắt.
- Hệ thống enemies và bẫy:
 - Nếu player bị enemies tấn công hoặc chạm vào các bẫy sẽ bị trừ thanh máu. Khi thanh máu về 0 sẽ chết và Respawn lại Checkpoint.
- Game Manager:
 - Có các điểm Checkpoint.
 - Khi kết thúc màn chơi sẽ chuyển sang màn chơi mới.
 - Khi player chết sẽ Respawn lại Checkpoint.
- Âm thanh:

- Nhạc nền Main Menu
- Nhạc nền mỗi màn chơi
- Hiệu ứng âm thanh khi combat



Hình 3. Màn hình Game Play

IV. KẾT LUẬN

1. Kết quả đạt được của đề tài

- Trình bày được tổng quan về công nghệ Unity Engine
- Hiểu rõ được tác dụng của Animation.
- Hiểu rõ được cách làm game trên Unity.
- Thiết kế các hoạt ảnh, nền, đồ họa cho game
- Xử lý được các lỗi cơ bản trong unity.
- Xử lý được âm thanh trong game.
- Xử lý được nhân vật trong game.
- Xử lý được các quái vật trong game.
- Xử lý được các vật phẩm trong game.
- Hoàn thành được game.
- Ứng dụng công nghệ Unity xử lý các bài toán xây dựng lên hệ thống trò chơi hiệu quả. Sử dụng ngôn ngữ C# lập trình lên ứng dụng.
- Xây dựng được trò chơi có tính giải trí cao và hiệu quả giúp người chơi thoải mái sau nhưng ngày làm việc mệt nhọc.
- Chức năng đơn giản dễ sử dụng phù hợp với mọi lứa tuổi.

2. Hạn chế của đề tài

- Chưa xử lý được tối ưu các ràng buộc, dữ liệu chưa được sắp xếp linh hoạt hợp lý.
- Còn nhiều chức năng chưa được hoàn thiện.
- Chưa bắt được hết các lỗi của hệ thống.
- Chưa xử lý được trạng thái hệ thống bị dừng khi đang thao tác và còn một số tồn tại trong việc đặt tên và sử dụng linh hoạt các điều khiển.
- Script vẫn hoạt động bình thường nhưng phát sinh một số lỗi không sửa được như:
 - Lỗi Player tương tác với một số góc bị sai
 - Lỗi Spikes Sendamage không khớp box
 - Lỗi Enemy respawn (hoạt ảnh và thanh máu)
 - Save game ko tối ưu, lỗi script dẫn đến tắt máy là mất data nên đã lược bỏ chức năng này.

3. Hướng phát triển của đề tài

- Tương tác được giữa người chơi thông qua hệ thống.
- Đồng bộ hóa dữ liệu giữa ứng dụng offline và hệ thống trực tuyến.
- Dữ liệu được tối ưu hóa đến mức chi tiết nhất.
- Tối ưu hóa nhân vật hơn giúp nhân vật chuyển được nhiều trạng thái nhân vật.

- Xây dựng, mở rộng thêm bản đồ giúp game trở nên đa dạng.
- Đặt quảng cáo để kiếm thu nhập.

TÀI LIỆU THAM KHẢO

- [1] <http://www.unity3dstudent.com>
- [2] <http://unity3d.com/learn>
- [3] Learn Unity3D Programming with UnityScript
- [4] <https://www.youtube.com/@MoonGameStudios>
- [5] <https://www.youtube.com/@PitiITNet>