BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**CHUYÊN ĐỀ TỐT NGHIỆP**

**XÂY DỰNG ỨNG DỤNG GAME ANDROID SỬ DỤNG UNITY**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn:** | **ThS.GV Huỳnh Tuấn Anh** |
| **Sinh viên thực hiện:** | **Nguyễn Hồng Quang** |
| **Mã số sinh viên:** | **61130921** |

Khánh Hòa – 2023

BỘ GIÁO DỤC VÀ ĐÀO TẠO

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**CHUYÊN ĐỀ TỐT NGHIỆP**

**XÂY DỰNG ỨNG DỤNG GAME ANDROID SỬ DỤNG UNITY**

|  |  |
| --- | --- |
| Giảng viên hướng dẫn: | ThS.GV Huỳnh Tuấn Anh |
| Sinh viên thực hiện: | Nguyễn Hồng Quang |
| Mã số sinh viên: | 61130921 |

Khánh Hòa – 2023

**TRƯỜNG ĐẠI HỌC NHA TRANG**

**Khoa/Viện:……………………………….**

**PHIẾU CHẤM ĐIỂM CHUYÊN ĐỀ TỐT NGHIỆP**

***(Dành cho cán bộ chấm điểm phản biện)***

Tên đề tài: Xây dựng ứng dụng game Android sử dụng Unity

Chuyên ngành: Kỹ thuật phần mềm

Họ và tên: Nguyễn Hồng Quang……………………………MSSV: 61130921

Người phản biện (học hàm, học vị, họ và tên):

Cơ quan công tác:

**I. Đánh giá và cho điểm của người phản biện (tính theo thang điểm 10)**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Tiêu chí đánh giá** | **Trọng số**  **(%)** | **Mô tả mức chất lượng** | | | | **Điểm** |
| **Giỏi** | **Khá** | **Đạt yêu cầu** | **Không đạt** |
| **9 - 10** | **7 - 8** | **5 - 6** | **< 5** |
| Hình thức bản thuyết minh | 30 |  |  |  |  |  |
| Nội dung bản  thuyết minh | 30 |  |  |  |  |  |
| Kết quả nghiên cứu | 20 |  |  |  |  |  |
| Mức độ trích dẫn và sao chép | 20 |  |  |  |  |  |
| **ĐIỂM TỔNG** | | | | | |  |

*Ghi chú: Làm tròn điểm tổng đến 1 số lẻ..*

**Kết luận:**

Đồng ý cho sinh viên: Được bảo vệ:  Không được bảo vệ: 

*Khánh Hòa, ngày…….tháng………năm………..* **Cán bộ chấm phản biện** *(Ký và ghi rõ họ tên)*

LỜI CAM ĐOAN

Em tên Nguyễn Hồng Quang, sinh viên khóa K61 thuộc lớp Công Nghệ Thông Tin 2 xin cam đoan rằng báo cáo chuyên đề này được viết bởi chính bản thân em dưới sự hướng dẫn của ThS.GV Huỳnh Tuấn Anh. Tất cả những thông tin được trình bày trong báo cáo này đều dựa vào quá trình nghiên cứu và tìm hiểu của bản thân. Toàn bộ các nguồn tài liệu và nguồn tham khảo được sử dụng trong báo cáo này đã được liệt kê và trích dẫn đầy đủ.

Em sẽ chịu trách nhiệm theo quy định của nhà trường cho lời cam đoan của bản thân nếu có vi phạm nào xảy ra.

LỜI CẢM ƠN

Em xin gửi lời cảm ơn chân thành đến ThS.GV Huỳnh Tuấn Anh vì đã hỗ trợ em thực hiện bài báo cáo này và cho em những lời khuyên bổ ích giúp em hoàn thiện đề tài này tốt nhất có thể.

Em cũng muốn gửi lời cảm ơn đến cộng đồng game Unity vì đã chia sẻ kiến thức và kinh nghiệm của họ, giúp em hoàn thiện game của mình.

Cuối cùng, em muốn gửi lời cảm ơn đến những độc giả của báo cáo này. Em hy vọng rằng báo cáo này sẽ cung cấp cho mọi người một số kiến thức về Unity mà em đã thực hiện trong quá trình phát triển và xây dựng game. Em mong nhận được những phản hồi và đóng góp ý kiến của mọi người để có thể cải thiện game của mình.

Em xin cảm ơn.

**MỤC LỤC**

[Chương 1. LỜI MỞ ĐẦU 1](#_Toc137581016)

[1. Mục tiêu và mục đích của việc làm game 1](#_Toc137581017)

[2. Cốt truyện và đặc tả các tính năng của game 1](#_Toc137581018)

[Chương 2. GIỚI THIỆU VỀ CÔNG NGHỆ VÀ CÔNG CỤ 3](#_Toc137581019)

[1. Sử dụng Unity Engine để phát triển game 3](#_Toc137581020)

[1.1. Khái niệm 3](#_Toc137581021)

[1.2. Các tính năng được cung cấp bởi Unity 3](#_Toc137581022)

[1.3. Cách thức hoạt động của Unity 4](#_Toc137581023)

[1.4. Các khái niệm quan trọng trong Unity 4](#_Toc137581024)

[2. Ngôn ngữ lập trình C# 5](#_Toc137581025)

[3. Các công cụ plugin được sử dụng để phát triển game 6](#_Toc137581026)

[Chương 3. THIẾT KẾ VÀ PHÁT TRIỂN GAME 7](#_Toc137581027)

[1. Các Scene trong game 7](#_Toc137581028)

[1.1. Main menu 7](#_Toc137581029)

[1.2. Level trong game 8](#_Toc137581030)

[2. Tag và Layer trong Unity 10](#_Toc137581031)

[2.1. Tag 10](#_Toc137581032)

[2.2. Layer 10](#_Toc137581033)

[3. Nhân vật chính (Player) 12](#_Toc137581034)

[4. Player di chuyển giữa các Scene 18](#_Toc137581035)

[5. Camera 19](#_Toc137581036)

[6. Prefab và các đối tượng bản sao 21](#_Toc137581037)

[6.1. Khái niệm cơ bản 21](#_Toc137581038)

[6.2. Các đối tượng có trong trong prefab 22](#_Toc137581039)

[7. Giao diện điều khiển (UI) 22](#_Toc137581040)

[7.1. Thiết kế giao diện 22](#_Toc137581041)

[7.2. Giao diện khi dừng game (Pause Menu) 23](#_Toc137581042)

[7.3. Cửa hàng mua vật phẩm (Shop) 24](#_Toc137581043)

[8. Âm thanh 24](#_Toc137581044)

[8.1. Khái niệm 24](#_Toc137581045)

[8.2. Thiết kế âm thanh 25](#_Toc137581046)

[9. Hệ thống lưu game 27](#_Toc137581047)

[10. Đối thoại với các NPC 28](#_Toc137581048)

[11. Nhân vật phản diện 30](#_Toc137581049)

[11.1. Slime (Chất nhầy) 31](#_Toc137581050)

[11.2. Bat (Dơi) 31](#_Toc137581051)

[11.3. Worm (Sâu) 32](#_Toc137581052)

[11.4. Frog (Ếch) 32](#_Toc137581053)

[12. Nhân vật phản diện là trùm (Boss) 33](#_Toc137581054)

[12.1. Golem (Quái vật đá) 35](#_Toc137581055)

[12.2. Swamp King (Vua đầm lầy) 36](#_Toc137581056)

[Chương 4. KẾT LUẬN 39](#_Toc137581057)

[1. Kết quả của việc làm game 39](#_Toc137581058)

[2. Những gì đã đạt được 39](#_Toc137581059)

[3. Hạn chế 39](#_Toc137581060)

[4. Kết luận và hướng phát triển trong tương lai 39](#_Toc137581061)

|  |  |  |
| --- | --- | --- |
| **Viết tắt** | **Đầy đủ** | **Ý nghĩa** |
| NPC | Non-playable character | Là các nhân vật trong game mà người chơi không thể điều khiển được. Các nhân vật do máy tính điều khiển để Player tương tác với chúng. |
| UI | User Interface | Giao diện người dùng được thiết kế để cung cấp cho người dùng một cách để tương tác và điều khiển các tính năng của ứng dụng. |
| BGM | Background Music | Nhạc nền được phát liên tục trong game |
| SFX | Sound Effect | Hiệu ứng âm thanh. |
| JSON | JavaScript Object Notation | Là một kiểu dữ liệu trong JavaScript sử dụng các cặp key – value (thuộc tính – giá trị), key là một chuỗi ký tự và value có thể là số, chuỗi, mảng,... Được sử dụng để truyền tải dữ liệu. |
| HP | Hit Point | Là một thuật ngữ trong game để chỉ số lượng máu hoặc điểm số mà một nhân vật hoặc đối tượng có thể chịu trước khi bị hạ gục |

**DANH MỤC CÁC TỪ VIẾT TẮT**

1. **LỜI MỞ ĐẦU**
2. **Mục tiêu và mục đích của việc làm game**

* **Mục tiêu**

Mục tiêu của việc làm game là để em có thể học hỏi các tính năng trong Unity và khả năng tương tác của nó. Việc làm game giúp em học hỏi một số tính năng cơ bản của Unity, bao gồm cách tạo và quản lý các đối tượng, cách thêm âm thanh và hiệu ứng, cách xử lý va chạm và tương tác với các đối tượng khác nhau,...

* **Mục đích**

Mục đích của việc làm game là để giúp em có được kiến thức nền tảng về Unity và khả năng tương tác của nó. Unity sẽ cung cấp một môi trường thực hành để em có thể áp dụng kiến thức của mình và thực hành các kỹ năng cần thiết để phát triển các trò chơi trong Unity.

Việc làm game cũng giúp em hiểu rõ hơn về quá trình phát triển game trong Unity, bao gồm các bước cơ bản để thiết kế, xây dựng và triển khai game. Được thực hành các kỹ năng này và tìm hiểu thêm về các tính năng nâng cao của Unity để phát triển các trò chơi phức tạp hơn trong tương lai.

1. **Cốt truyện và đặc tả các tính năng của game**

* **Cốt truyện**

Cốt truyện của game xoay quanh cuộc phiêu lưu của nhân vật chính là một chú cáo, được truyền lại sức mạnh từ trưởng làng cậu bắt đầu chuyến hành trình của mình đánh bại kẻ thù xâm lược đến ngôi làng của cậu từ vùng đất đầm lầy. Chú cáo nhỏ sẽ lên đường đánh đuổi kẻ thù và những con trùm hùng mạnh để trả lại bình yên cho làng của mình.

* **Kịch bản hoạt động của game**

Người chơi sẽ điều khiển nhân vật chính để khám phá các khu vực, đánh bại quái vật và thu thập các vật phẩm. Tương tác với các NPC để có thể theo dõi được cốt truyện hoặc lắng nghe thông tin. Người chơi có thể mua vật phẩm từ cửa hàng để sử dụng trong quá trình chơi của mình. Người chơi sẽ phải sử dụng kỹ năng và vật phẩm của mình để vượt qua các thử thách trong các màn chơi và mở ra các khu vực mới.

* **Các tính năng chính**

**Khám phá thế giới:** Người chơi sẽ điều khiển nhân vật chính vượt qua các thử thách để di chuyển đến khu vực tiếp theo.

**Tương tác với NPC:** Trong game sẽ có các NPC để người chơi có thể tương tác như nói chuyện, hỏi thông tin,...

**Vượt các cạm bẫy:** Trên đường đi người chơi phải vượt qua các cạm bẫy một cách an toàn để không bị mất máu hoặc bị hạ gục.

**Đánh bại kẻ thù:** Mỗi màn chơi sẽ có những con quái trên đường đi, chúng có cách di chuyển và tấn công khác nhau nhằm tạo thử thách cho người chơi.

**Thu thập vật phẩm:** Vật phẩm có thể rớt ra khi đánh bại quái vật hoặc phá hủy các vật có thể phá hủy trên đường.

**Mua vật phẩm từ cửa hàng:** Người chơi có thể mua vật phẩm trong cửa hàng bằng tiền kiếm được.

Các tính năng của đó giúp tạo ra sự kết nối giữa người chơi và trò chơi, tạo ra một trải nghiệm chơi game tốt nhất cho họ. giúp tạo ra sự cạnh tranh và thử thách giúp người chơi tiếp cận và phát triển, học hỏi các kỹ năng mới trong quá trình chơi game.

1. **GIỚI THIỆU VỀ CÔNG NGHỆ VÀ CÔNG CỤ**
2. **Sử dụng Unity Engine để phát triển game**
   1. **Khái niệm**

Unity Engine là công cụ miễn phí dùng để phát triển các trò chơi đa nền tảng. Được phát triển bởi “Unity Technologies” và bắt đầu được khởi động vào năm 2005. Unity Engine cung cấp các công cụ và tính năng để giúp các nhà làm game có thể tạo ra các trò chơi chất lượng cao. Với tính linh hoạt của nó, các nhà lập trình game có thể tạo ra các trò chơi và ứng dụng trên đa nền tảng phổ biến nhất gồm: PC, laptop, điện thoại thông minh và các dòng máy chơi game,...

Unity đã và đang được sử dụng rộng rãi trong ngành công nghiệp game, được sử dụng để tạo ra những game nổi tiếng hiện nay như: Genshin Impact của miHoYo, Tốc chiến của nhà phát hành Riot Games, Hollow Knight của Team Cherry,... đều sử dụng Unity Engine để xây dựng.

* 1. **Các tính năng được cung cấp bởi Unity**

Unity Engine cung cấp một loạt các tính năng và công cụ để giúp các nhà phát triển tạo ra các trò chơi chất lượng cao. Các tính năng này bao gồm:

* **Đồ họa:** Unity Engine có khả năng tạo ra các đồ họa 2D và 3D đẹp mắt và chuyển động mượt mà.
* **Giao diện UI:** Unity Engine tạo ra các giao diện UI cho hệ thống như text box, button, drop bar,...
* **Networking:** Hỗ trợ chế độ chơi mạng, trực tuyến, nhiều người chơi,...
* **Vật lý:** Unity Engine hỗ trợ tính toán vật lý chính xác để tạo ra các hiệu ứng và tương tác thực tế trong trò chơi.
* **Âm thanh:** Unity Engine có khả năng tạo ra các hiệu ứng âm thanh chân thực và đa dạng để tăng cường trải nghiệm của người chơi.
* **Quản lý tài nguyên:** Unity Engine cho phép các nhà phát triển quản lý tài nguyên của mình, bao gồm các đối tượng, âm thanh, hình ảnh và vật liệu.
* **Công cụ phát triển:** Unity Engine cung cấp các công cụ phát triển cho phép các nhà phát triển tạo ra các trò chơi một cách dễ dàng và nhanh chóng, bao gồm các trình chỉnh sửa đồ họa, trình chỉnh sửa vật lý và trình soạn thảo mã nguồn.
* **Scripting:** Unity hỗ trợ các ngôn ngữ lập trình như C# và JavaScript để phát triển trò chơi.
  1. **Cách thức hoạt động của Unity**

Unity hoạt động bằng cách sử dụng một hệ thống thành phần (component system). Trong hệ thống này, các đối tượng trong trò chơi được xây dựng từ các thành phần khác nhau như đồ họa, vật lý và âm thanh. Các thành phần này có thể được kết hợp để tạo ra các đối tượng phức tạp trong trò chơi.

Các nhà làm game sử dụng Unity Editor để tạo ra các đối tượng và môi trường trong trò chơi. Họ cũng có thể viết các mã nguồn hoặc thêm các thành phần để tạo ra các hành động và tương tác giữa các đối tượng trong trò chơi.

Khi trò chơi được xây dựng xong, các nhà làm game có thể xuất trò chơi sang nhiều nền tảng khác nhau.

* 1. **Các khái niệm quan trọng trong Unity**

**GameObject:** Là đối tượng cơ bản trong Unity, đại diện cho một đối tượng có trong game. Mỗi đối tượng có thể có nhiều thành phần (component) khác nhau để cung cấp các tính năng cho đối tượng đó như: các mã nguồn (Script), Rigibody, Collider,...

**Component:** Là các thành phần được thêm vào một đối tượng như, script, đồ họa, vật lý, âm thanh,... Các thành phần này được thêm vào để mở rộng các tính năng cho đối tượng đó.

**Script:** Là mã nguồn được viết bằng các ngôn ngữ lập trình như C# hoặc JavaScript để tạo ra các hành động và tương tác trong game cho đối tượng.

**Prefab:** Là đối tượng được tạo ra từ một đối tượng gốc và các thành phần của nó. Prefab có thể được sử dụng để tạo ra nhiều đối tượng bản sao từ một đối tượng gốc trong game.

**Scene:** Là môi trường trong trò chơi, chứa các đối tượng môi trường và các thành phần của chúng. Các scene có thể được sử dụng để tạo ra các màn chơi khác nhau trong trò chơi.

**ScriptableObject:** Là một lớp đối tượng trong Unity, cho phép các nhà làm game tạo ra các đối tượng dữ liệu có thể được sử dụng trong trò chơi. ScriptableObject có thể được sử dụng để lưu trữ các dữ liệu như cấu hình trò chơi, dữ liệu game và các tài nguyên khác.

**Asset:** Là tài nguyên trong Unity, bao gồm các file đồ họa, âm thanh, vật liệu và các thành phần khác. Các asset có thể được sử dụng để tạo ra các đối tượng và môi trường trong game.

**Inspector:** Là một giao diện đồ họa trong Unity, cho phép nhà làm game chỉnh sửa các thuộc tính của các đối tượng và các thành phần của chúng.

**Transform:** Là thành phần luôn luôn có của mỗi đối tượng trong Unity, đại diện cho vị trí tọa độ, góc quay và tỷ lệ kích thước của đối tượng đó trong không gian vector ba chiều.

**Rigidbody:** Là thành phần trong Unity, mô phỏng các chức năng vật lý. Rigidbody có thể được sử dụng để xử lý các chuyển động và tương tác vật lý trong trò chơi.

**Collision:** Là sự va chạm giữa hai đối tượng trong trò chơi. Collision có thể được xử lý bằng cách sử dụng các thành phần như Collider và Rigidbody để xác định các hành động và tương tác khi hai đối tượng va chạm với nhau.

**Collider:** Là thành phần trong Unity, đại diện cho hình dạng của một GameObject trong không gian 3D. Collider được sử dụng để xác định các vùng không gian mà GameObject có thể va chạm với các đối tượng khác trong trò chơi.

**Trigger:** Là một loại Collider trong Unity, được sử dụng để xác định các vùng kích hoạt trong trò chơi. Khi một GameObject đi qua một Trigger, nó có thể kích hoạt các hành động hoặc tương tác khác trong trò chơi.

1. **Ngôn ngữ lập trình C#**

C# là một ngôn ngữ lập trình đa nền tảng được phát triển bởi Microsoft. Nó được thiết kế để xây dựng các ứng dụng Windows, ứng dụng web và ứng dụng di động. C# là một ngôn ngữ lập trình hiện đại, có cú pháp đơn giản và dễ hiểu, hỗ trợ tính đa hình, kế thừa và đa luồng, giúp cho việc phát triển ứng dụng trở nên dễ dàng và nhanh chóng hơn. Ngoài ra, C# còn được tích hợp với .NET Framework, một nền tảng phát triển ứng dụng mạnh mẽ của Microsoft, giúp cho việc phát triển ứng dụng trở nên linh hoạt và tiện lợi hơn.

1. **Các công cụ plugin được sử dụng để phát triển game**
2. **Unity Asset Store**

Unity Asset Store là một cửa hàng trực tuyến chứa nhiều tài nguyên và công cụ hữu ích cho những nhà lập trình game. Họ có thể tìm kiếm các gói tài nguyên do các người dùng khác đăng lên để họ thêm vào dự án của mình. Unity Asset Store có nhiều nguồn tài nguyên được cung cấp cho người dùng từ các model nhân vật đến các công cụ khác hỗ trợ người dùng trong quá trình phát triển game của họ. Các gói tài nguyên trong Unity có thể bắt người dùng phải trả phí tuy nhiên cũng có nhiều gói ở mức miễn phí mà có chất lượng cao.

1. **TextMesh Pro**

TextMesh Pro là một plugin được tích hợp sẵn trong Unity, nó cung cấp các tính năng hỗ trợ người dùng trong việc tạo ra các văn bản cho game của họ, với các tính năng khác của nó như: Định dạng văn bản, tạo hiệu ứng chuyển động cho văn bản, chuyển đổi văn bản, hiển thị văn bản trong game,... Nó cũng hỗ trợ đa ngôn ngữ và các font chữ khác nhau có thể được thêm vào bởi người dùng, giúp tạo ra game đa ngôn ngữ dễ dàng hơn.

1. **Ink**

Được phát triển bởi công ty “Inkle Studios”, nó là một ngôn ngữ lập trình mã nguồn mở cho phép những nhà lập trình game tạo ra các cốt truyện độc đáo trong game của họ, với các lựa chọn và hướng đi khác nhau tùy vào lựa chọn của người chơi. Ink có thể được sử dụng với Unity, điều này giúp tạo ra các câu truyện trong game một cách dễ dàng hơn.

1. **Cinemachine**

Cinemachine là một plugin mở rộng cho camera của Unity, nó cung cấp các tính năng đa dạng và chất lượng cao trong việc tạo ra các cảnh quay trong game với những tính năng như: Tạo các cảnh quay tự động, giúp camera theo một đối tượng nào đó trong game, không cho phép camera đi ra khỏi vùng nào đó trong game,... Cienmachine là một phiên bản mở rộng độc đáo của camera thực hiện các chức năng mà một camera đơn thuần của Unity không thể làm và với các chức năng có sẵn trong nó người dùng có thể tạo ra các cảnh quay đa dạng hơn mà không cần phải lập trình các chức năng cho camera.

1. **THIẾT KẾ VÀ PHÁT TRIỂN GAME**

Game được lấy cảm hứng từ trò chơi “mario cổ điển” kết hợp với tựa game “Hollow Knight” bao gồm một thế giới rộng lớn được chia thành nhiều khu vực khác nhau, có các vật phẩm để thu thập và các kẻ thù để đánh bại. Tuy nhiên, game cũng có các yếu tố thử thách khác nhau giữa các khu vực để tạo cảm giác mới lạ cho người chơi.

1. **Các Scene trong game**
   1. **Main menu**

Main menu là một scene trong trò chơi, được sử dụng để hiển thị các tùy chọn và chức năng chính của trò chơi. Main menu thường được hiển thị khi khởi động trò chơi hoặc khi người chơi muốn quay lại menu trong quá trình chơi.



* **Main menu bao gồm các thành phần sau:**

**Background:** Là hình ảnh hoặc video được sử dụng làm nền cho main menu. Background được thiết kế để phù hợp với chủ đề của trò chơi.

**Logo:** Là biểu tượng của trò chơi, được hiển thị ở góc trên cùng của main menu. Logo giúp người chơi nhận ra trò chơi và thông tin cơ bản của nó.

**Các tùy chọn:** Là các nút hoặc menu tạo ra từ UI của Unity được sử dụng để hiển thị các tùy chọn và chức năng chính của trò chơi. Các tùy chọn bao gồm "New Game" để bắt đầu một trò chơi mới, "Continute" để tiếp tục game nếu tồn tại dữ liệu game, "Quit" để thoát game.

* **Chức năng của các nút trong menu.**

Đầu tiên nút “NEW GAME” được dùng để tạo ra dữ liệu mới cho game và ghi đè lên dữ liệu cũ nếu có để tạo mới game cho người chơi.

Nút “CONTINUTE” được dùng để tải lại game cho người chơi. Nếu không tồn tại dữ liệu game để đọc thì nút sẽ bị vô hiệu hóa để người chơi không thể bấm được.

Nút “QUIT” để thoát game.

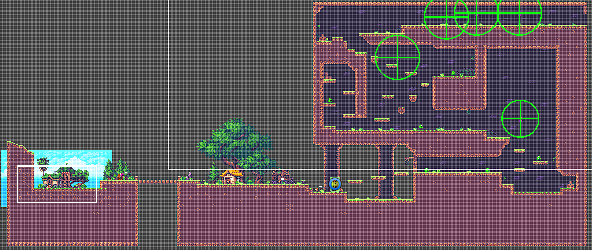
* 1. **Level trong game**

Level là một phần của trò chơi, thường được lập trình viên thiết kế từ các Scene để hiển thị một cấp độ hoặc màn chơi trong trò chơi. Mỗi level thường có một mục tiêu hoặc nhiệm vụ cụ thể mà Player cần vượt qua để có thể đi tiếp.

**Một Scene được cấu thành bởi:**

**Tilemap:** Được sử dụng để tạo ra các môi trường và địa hình cho game 2D. Tilemap bao gồm một lưới các ô vuông (tile) được sắp xếp theo hàng và cột, mỗi ô vuông đại diện cho một phần của môi trường hoặc địa hình trong trò chơi. Các ô trong tilemap có thể được thiết kế với nhiều kiểu dáng khác nhau. Nó được sử dụng để tạo ra một môi trường 2D trong trò chơi, giúp người chơi có thể di chuyển nhân vật và tương tác trong môi trường đó.

Sử dụng Tilemap là một cách tiết kiệm thời gian và tối ưu hóa quá trình thiết kế môi trường trong trò chơi.



**Các component được sử dụng để xây dựng Tile map:**

Tilemap có Tag và Layer là “Ground” để các đối tượng khác kiểm tra có đang ở trên mặt đất hay không.

* **Tilemap:** Là một thành phần của Unity được sử dụng để tạo ra các bản đồ tile-based (dựa trên các ô vuông) cho môi trường 2D trong trò chơi. Tilemap cũng cho phép người dùng dễ dàng chỉnh sửa và thay đổi các tile trong môi trường của họ.
* **Tilemap Renderer:** Là một thành phần khác của Unity được sử dụng để hiển thị Tilemap trong trò chơi.
* **Rigibody 2D:** Tilemap sử dụng Body Type “Static” trong Rigidbody2D, đối tượng sẽ không di chuyển hoặc tương tác với các đối tượng khác trong môi trường.
* **Tilemap Collider 2D:** Là một thành phần của Unity được sử dụng để tạo ra các vùng va chạm (collision) cho Tilemap trong môi trường 2D của trò chơi. Các vùng va chạm này có thể được tùy chỉnh cho phù hợp với yêu cầu của nhà lập trình game.
* **Composite Collider 2D:** Là một Collider khác của Unity, khi không sử dụng Composite Collider 2D thì các vùng va chạm trong Tilemap Collider 2D là mỗi ô vuông của tile map điều này sẽ gây ảnh hưởng khi Player di chuyển, vì vậy khi sử dụng Composite Collider 2D nó bao bọc một lớp bên ngoài của tile lớn thay vì biến các ô trong tile thành mỗi collider độc lập.

**Môi trường:** Là các đối tượng và vật phẩm xung quanh nhân vật trong trò chơi. Môi trường có thể bao gồm các đối tượng tĩnh như cây cối, đá, tường,... và một số đối tượng tương tác được.

* Các đối tượng không tương tác được dùng để làm đẹp cho cảnh vật.



* Các đối tượng tương tác được như thùng gỗ sẽ rớt ra vật phẩm khi bị người chơi phá hủy.



1. **Tag và Layer trong Unity**
   1. **Tag**

Tag trong Unity là một cách để phân loại các đối tượng trong trò chơi. Mỗi đối tượng trong trò chơi có thể được gán vào một Tag cụ thể, và các đối tượng trong cùng một Tag sẽ có các thuộc tính và hành vi tương tự nhau.

**Các tính năng của Tag trong Unity bao gồm:**

* **Tìm kiếm:** Tìm kiếm đối tượng theo Tag sử dụng các chức năng tìm kiếm của Unity. Điều này giúp giảm thiểu thời gian tìm kiếm và tăng hiệu suất của trò chơi.
* **Xử lý sự kiện:** Các sự kiện có thể được xử lý bằng Tag của các đối tượng sử dụng các phương thức và câu lệnh xử lý của Unity. Điều này giảm thiểu số mã lệnh cần được viết và tăng hiệu suất của game.

Các Tag trong Unity được tạo ra thông qua cửa sổ Inspector hoặc bằng cách sử dụng mã lệnh trong script. Các Tag được gán cho các đối tượng sử dụng thuộc tính Tag của các thành phần Collider hoặc Renderer.

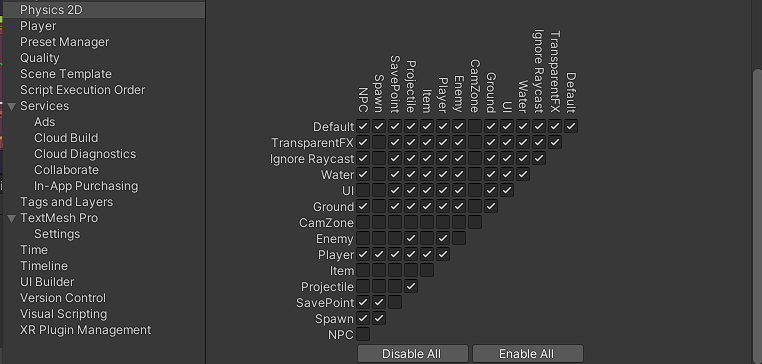
* 1. **Layer**

Layer trong Unity là một cách để phân loại các đối tượng trong trò chơi. Mỗi đối tượng trong trò chơi có thể được gán vào một Layer cụ thể mặc định sẽ là “Default”, và các đối tượng trong cùng một Layer sẽ tương tác với nhau theo cách được xác định bởi các thiết lập Layer.

**Các tính năng của Layer trong Unity bao gồm:**

* ***Collision Detection:*** Các đối tượng trong cùng một Layer sẽ tương tác với nhau khi xảy ra va chạm. Các đối tượng trong các Layer khác nhau sẽ không tương tác với nhau.
* ***Raycasting:*** Raycasting là một kỹ thuật được sử dụng để xác định vị trí của các đối tượng trong trò chơi. Khi sử dụng Raycasting, có thể chỉ định các Layer mà Raycast sẽ tương tác với, giúp giảm thiểu thời gian tính toán và tăng hiệu suất của trò chơi.
* ***Rendering:*** Các đối tượng trong cùng một Layer sẽ được vẽ trên cùng một lớp khi hiển thị trên màn hình. Điều này giúp tăng hiệu suất của trò chơi bằng cách giảm số lượng lệnh vẽ cần được thực thi.
* ***Sorting Order:*** Các đối tượng trong cùng một Layer có thể được sắp xếp theo thứ tự hiển thị trên màn hình. Điều này giúp đảm bảo rằng các đối tượng sẽ được vẽ đúng thứ tự và tránh việc che phủ lẫn nhau.

Để các đối tượng không va chạm với nhau, sử dụng tính năng Layer Collision Matrix trong Unity. Mở cửa sổ Edit và chọn Project Settings > Physics 2D. Tìm kiếm mục Layer Collision Matrix sẽ thấy danh sách các Layer trong trò chơi. Có thể bật hoặc tắt tính năng va chạm giữa các Layer bằng cách chọn và tắt các checkbox tương ứng. Để ngăn các đối tượng trong Layer va chạm với nhau thì tắt tính năng va chạm giữa các Layer đó. Ví dụ, nếu muốn các đối tượng trong Layer "Enemy" không va chạm với nhau thì tắt tính năng va chạm giữa "Enemy" và "Enemy”. Sau khi đã tắt tính năng va chạm giữa các Layer tương ứng, các đối tượng trong các Layer đó sẽ không còn va chạm với nhau nữa.



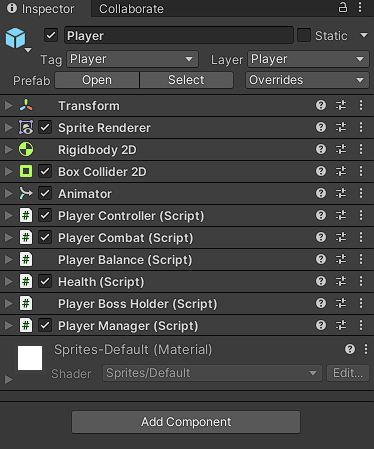
1. **Nhân vật chính (Player)**

****

Nhân vật chính có ngoại hình là là một chú cáo. Tên đối tượng là “Player”. Player có thể di chuyển, nhảy và bắn ra các viên đạn lửa.

Lượng máu khởi đầu: 4

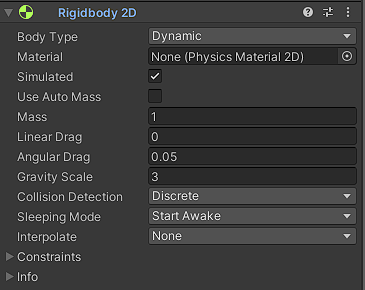
Phương thức tấn công: Phóng ra những quả cầu lửa.

Player có các Component như sau:

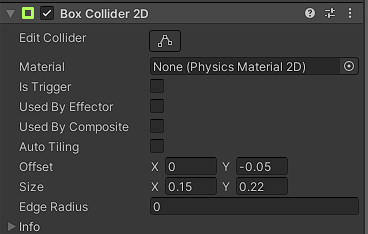
Nhân vật chính có Tag và Layer là “Player”.

**Sprite Renderer:** Sprite Renderer là một thành phần của Unity được sử dụng để hiển thị các sprite (hình ảnh 2D) trong môi trường 2D của trò chơi. Nó cho phép người dùng tùy chỉnh các thuộc tính của sprite, bao gồm kích thước, vị trí, độ trong suốt,…

**Rigibody 2D:** Thực hiện xử lý vật lý cho Player ở đây sử dụng Body Type là “Dynamic” được thiết kế để di chuyển theo mô phỏng. Nó có đầy đủ các thuộc tính sẵn có như khối lượng (Mass) và lực cản (Linear Drag), đồng thời bị ảnh hưởng bởi lực hấp dẫn (Gravity) và các lực khác. Dynamic sẽ tương tác với mọi Body Type khác và là Body Type tương tác nhiều nhất trong các loại Body Type. Dynamic luôn là mặc định cho một Rigidbody 2D, bởi vì nó là loại Body Type phổ biến nhất cho những đối tượng cần di chuyển. Đây cũng là loại Body Type hiệu suất cao nhất, vì bản chất năng động và khả năng tương tác với mọi thứ xung quanh. Tất cả các thuộc tính Rigidbody 2D đều khả dụng với loại Body Type này.

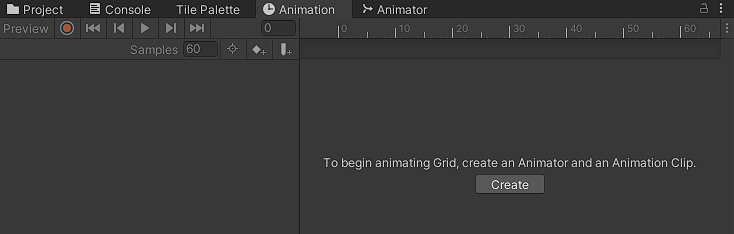


Tại Rigibody2D của Player set Khối lượng là “1” và Trọng lực là “3” và giữ nguyên các thành phần còn lại. Việc thay đổi Khối lượng sẽ làm cho đối tượng nhẹ hoặc nặng hơn và thay đổi Trọng lực sẽ làm cho đối tượng ảnh hưởng bởi trọng lực, nếu trọng lượng bằng 0 thì đối tượng sẽ không bị ảnh hưởng bởi trọng lực và nếu tăng dần lên thì trọng lực sẽ càng mạnh thì Player rơi xuống càng nhanh theo mô phỏng trọng lực.

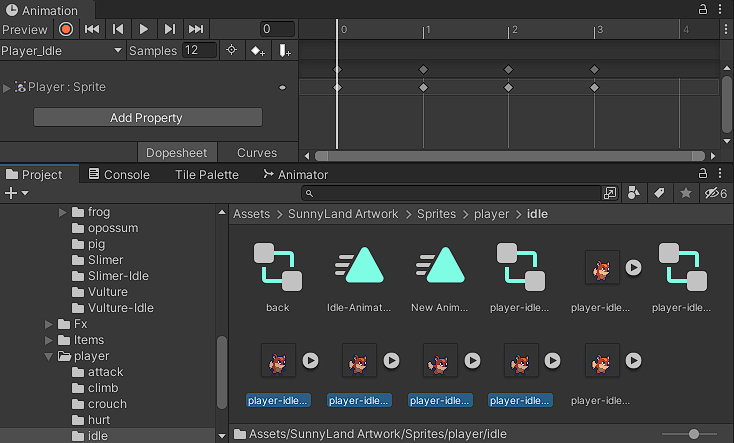
**Box Collider 2D:** Sử dụng để xác định các vùng không gian mà có thể va chạm với các đối tượng khác trong trò chơi với Player. Có thể điều chỉnh được kích thước của Box Collider 2D sao cho phù hợp với đối tượng trong bảng điều khiển sử dụng Size và Offset. Box Collider sẽ hiển thị là hình hộp màu xanh lá để xác định vị trí va chạm. Việc sử dụng Box Collider 2D giúp Player không bị rơi khỏi map vì có Collider của Tile Map tương tác với Collider của Player.

**Animator:** Tạo ra các hiệu ứng hoạt ảnh cho đối tượng cho nhân vật chính từ Animation. Cách sử dụng Animator và Animation:

1. Sử dụng Animator: Bắt đầu bằng cách tạo một Animator mới trong Unity. Chọn Window > Animation > Animator để tạo một Animator mới. Và Window > Animation > Animation sau đó nhấn “Create” để tạo Animation mới.



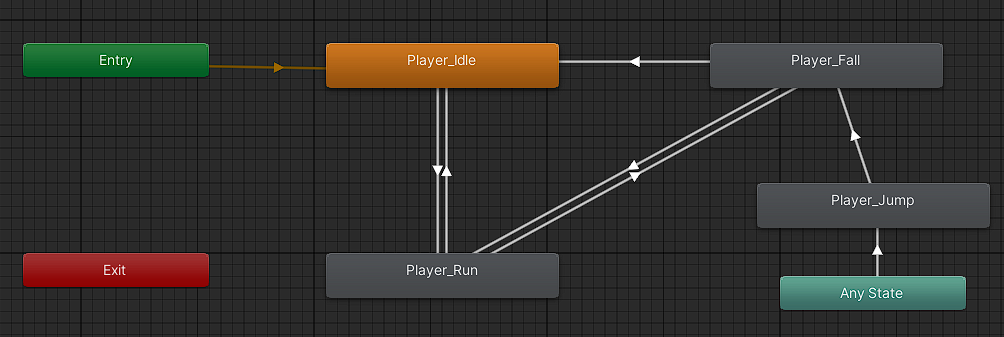
1. Tạo hoạt ảnh cho đối tượng: Sau khi tạo Animator, có thể thêm các Animation vào Animator bằng cách kéo và thả các Animation vào Animator. Cũng có thể tạo Animation mới bằng cách chọn Create New Animation trong Animator, sau đó kéo và thả các sprite cần tạo Animation vào nếu Animation chạy quá nhanh thì có thể thay đổi tốc độ bằng cách chỉnh sửa số bên cạnh nút “Samples”, con số đó có ý nghĩa là bao nhiêu khung hình trên giây mặc định là 60 nghĩa là 60fps.



1. Tạo State: State trong Animator là một trạng thái logic của animation clip trong Animator. Trong Animator có các State và được liên kết với nhau thông qua Transitions. Tạo các State mới bằng cách chọn “Add State” trong Animator.



1. Tạo Transition: Transition là cách chuyển đổi giữa các State trong Animator. Tạo các Transition mới bằng cách kéo và thả các State vào Animator và kết nối chúng với nhau bằng cách click chuột phải từ Animation này và click chuột trái để nối đến Animation cần nối.



**Trong các Transition có các chức năng sau:**

* ***Duration:*** Đây là thời gian Transition sẽ mất để chuyển đổi từ State hiện tại sang State tiếp theo.
* ***Exit Time:*** Đây là thời gian Transition sẽ kết thúc và chuyển đổi sang State tiếp theo. Nếu Exit Time được đặt là 1, Transition sẽ kết thúc khi State kết thúc.
* ***Offset:*** Đây là thời gian Transition sẽ bắt đầu sau khi State hiện tại bắt đầu. Nếu Offset được đặt là 0, Transition sẽ bắt đầu ngay khi State hiện tại bắt đầu.
* ***Has Exit Time:*** Đây là cờ để xác định liệu Transition có thể kết thúc trước khi State hiện tại kết thúc hay không. Nếu Has Exit Time được đặt là false, Transition sẽ kết thúc khi điều kiện được đáp ứng.
* ***Conditions:*** Điều kiện là các biểu thức logic, xác định khi nào Transition được kích hoạt. Ví dụ, có thể sử dụng điều kiện để xác định khi nào Player nhấn nút để chuyển đổi sang State tiếp theo.

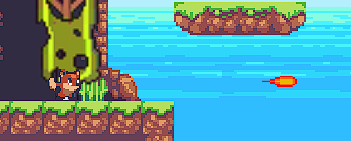
**Các script được dùng để xây dựng đối tượng Player:**

* **Player Controller:** Điều khiển hành động và chuyển động của Player.

Chứa các phương thức xác định hướng di chuyển của Player, tốc độ di chuyển, lực nhảy của Player, và trạng thái di chuyển của Player để thực hiện animator khi Player di chuyển. Khi Player nhấn nút di chuyển, nhảy trên giao diện điều khiển thì Player sẽ thực hiện các chức năng di chuyển được tạo trong script và thay đổi hoạt ảnh chuyển động phù hợp với các trạng thái di chuyển của Player.

* **Player Combat:** Được sử dụng để điều khiển hành động tấn công của Player.

Chứa các phương thức bao gồm thời gian hồi chiêu giữa các lần tấn công của Player, trạng thái tấn công của Player, đối tượng Projectile từ Prefab để tạo ra đối tượng tấn công, và vị trí để bắn viên đạn. Khi người chơi giữ nút tấn công thì Player sẽ thực hiện tấn công liên tục và khi người chơi thả tay ra thì Player cũng dừng tấn công.



* **Player Balance:**Quản lý số tiền và lượng thuốc hồi máu của Player.

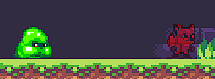
Chứa các phương thức để đếm số tiền của Player và số lượng thuốc hồi máu hiện tại của Player. Khi Player va chạm với một đối tượng có tag "currency". Phương thức sẽ tăng giá trị tiền lên 1. Nếu Player đủ tiền và mua được bình máu khi đó số lượng bình máu của Player sẽ tăng lên một.

* **Health:** Được sử dụng để quản lý HP và các chức năng liên quan.

Chứa các phương thức để quản lý số lượng máu hiện tại của Player và số lượng máu tối đa của Player. Các UI đại diện cho các trạng thái máu còn lại và máu bị mất của Player trên giao diện hiển thị. Lượng máu hiện tại của Player không lớn hợn lượng máu tối đa. Khi Player bị nhận sát thương, nếu lượng máu của Player lớn hơn 0, lượng máu hiện tại của Player sẽ bị trừ đi một và Player sẽ bất tử trong một khoảng thời gian ngắn.



Trong hoảng thời gian bất tử của Player sau khi bị tấn công, Player sẽ nhấp nháy màu đỏ và bị mất kiểm soát một khoảng thời gian ngắn. Layer “Player” sẽ không tương các với Layer “Enemy” của quái vật để Player không nhận thêm sát thương, sau khi thời gian bất tử kết thúc thì hai Layer có thể tương tác được với nhau.

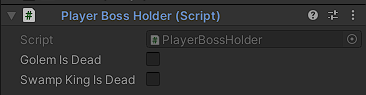


Khi lượng máu của Player nhỏ hơn hoặc bằng 0. Player sẽ được dịch chuyển đến điểm hồi sinh (Checkpoint). Lượng máu của Player sẽ được khôi phục đến lượng máu tối đa. Điểm hồi sinh của Player là những lá cờ chứa thông tin về tọa độ và màn chơi để Player có thể thực hiện chức năng hồi sinh.



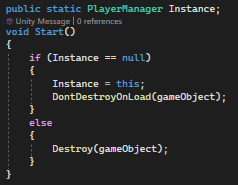
* **Player Boss Holder*:*** Được sử dụng để quản lý những con trùm.

Ban đầu những con trùm chưa bị đánh bại thì nó sẽ mặc định là false. Khi một con trùm bị đánh bại thì dữ liệu của nó sẽ được gán thành true. Những con trùm chưa bị đánh bại thì sẽ được gọi để thực hiện phương thức triệu hồi, nếu đã bị đánh bại thì sẽ không triệu hồi chúng nữa.



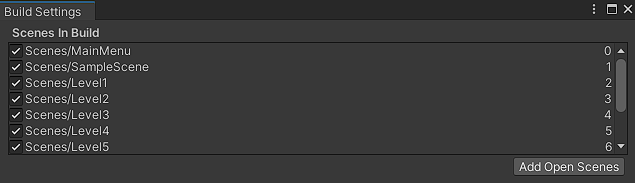
* **Player Manager:**Được sử dụng để đối tượng Player không bị hủy khi di chuyển từ Scene này sang Scene khác.

Nếu Player đến Scene khác mà ở đó không có tồn tại đối tượng Player thì Player sẽ bị hủy khi đó sử dụng phương thức DontDestroyOnLoad() là phương pháp làm cho một đối tượng không bị hủy khi chuyển qua Scene mới. Nếu Scene mới có tồn tại Player thì sẽ hủy Player hiện tại để tránh tình trạng tồn tại hai Player cùng một lúc.

******

1. **Player di chuyển giữa các Scene**

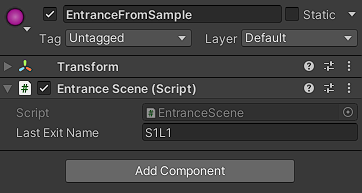
Player có khả năng di chuyển qua lại giữa các scene khi chạm vào đối tượng “Exit” để đi từ Scene hiện tại đến Scene kế. Đầu tiên cần phải thực hiện bỏ Scene vào Build bằng cách chọn File > Build Settings... > tìm ô Scenes In Build ở trên cùng sau đó thêm các Scene đã tạo vào trong ô theo thứ tự, lúc này Player mới có thể di chuyển qua lại giữa các Scene được.

******

Script “ExitScene” được gắn vào đối tượng “Exit” có Collider 2D, script chứa hai thông số là “sceneToLoad” và “exitName” lần lượt là Scene cần được tải và tên của vị trí mà Player sẽ đi ra. Phương thức “OnCollisionEnter2D” được gọi khi có va chạm giữa “Exit” và đối tượng khác có tag là "Player". Khi xảy ra va chạm, script sẽ lưu trữ tên của “exitName” hiện tại vào PlayerPrefs và tải Scene cùng tên.



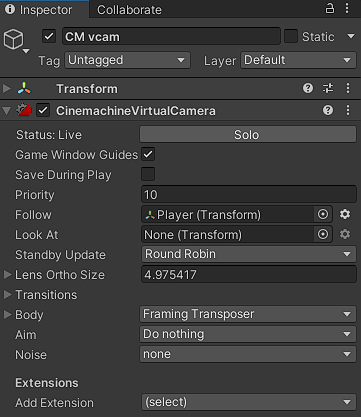
Script “EntranceScene” được gắn vào đối tượng “Entrance” ở một Scene khác. Nếu tên “exitName” mà Player đã đi qua trùng với “lastExitName” được lưu trữ thì script sẽ dịch chuyển Player đến vị trí của đối tượng “EntranceScene” cùng tên.

******

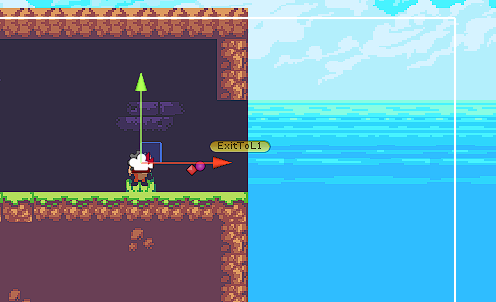
1. **Camera**

Camera trong Unity là một thành phần tạo ra hình ảnh của một góc nhìn cụ thể trong Scene với đầu ra được hiển thị lên màn hình hoặc được chụp dưới dạng texture. Cinemachine là một công cụ quản lý camera trong game, Cinemachine cũng hỗ trợ nhiều loại camera như camera bám theo đối tượng, camera xoay quanh đối tượng, camera di chuyển tự do,...

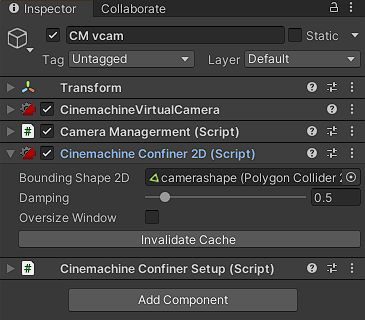
Để Camera tập trung và di chuyển theo Player, đầu tiên tìm đối tượng “CM vcam” đây là Cinemachine được tích hợp với Camera gốc có sẵn trong Unity, và mở phần “CinemachineVirtualCamera” sau đó kéo và thả đối tượng “Player” vào trường “Follow”, khi đó Camera sẽ tự động di chuyển theo Player.



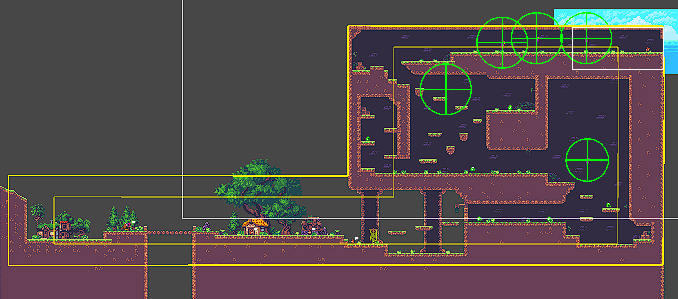
Để Camera không chiếu những cảnh bên ngoài rìa của bản đồ thì có thể thực hiện như sau:

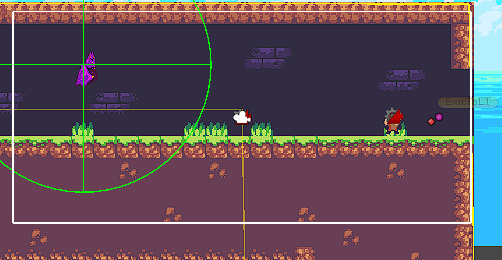


1. Đầu tiên tạo một đối tượng tên là “camerashape” và thêm Polygon Collider 2D cho nó, sau đó thêm layer tên “CamZone” vào đối tượng và lưu ý chỉnh sửa layer của CamZone sao cho nó không va chạm với bất kỳ layer nào khác nếu không nó sẽ đẩy tất cả đối tượng khác có Collider ra ngoài.
2. Tiếp theo điều chỉnh hình dạng của Polygon Collider 2D sao cho nó bao quanh khu vực không muốn để cho Camera lọt ra ngoài được.
3. Cuối cùng mở “CM vcam” và tìm “Cinemachine Confiner 2D” và thêm đối tượng “camerashape” vào ô Bounding Shape 2D.



Khi đó khu vực của Polygon Collider 2D sẽ trở thành màu vàng và lúc này Camera sẽ không thể ra khỏi khu vực đã được bao phủ bởi “camerashape”.





1. **Prefab và các đối tượng bản sao**
   1. **Khái niệm cơ bản**

Trong Unity, Prefab được sử dụng để lưu trữ và tái sử dụng các đối tượng trong trò chơi. Prefab có thể chứa một hoặc nhiều đối tượng được lưu trong Assets của dự án. Prefab có thể chứa các thành phần như Rigibody, Textures, Scripts, Colliders, Audio, Animations,... để tạo ra các hiệu ứng và tính năng.

Các đối tượng trong Prefab được sắp xếp và bố trí theo nhu cầu của trò chơi, và có thể được sử dụng lại trong nhiều Scene khác nhau trong dự án. Có thể tạo các đối tượng bản sao từ Prefab vào Scene bằng cách kéo thả hoặc gọi nó bằng cách sử dụng các câu lệnh trong script.

Khi thay đổi thông số của đối tượng trong Prefab cũng thay đổi thông số của các đối tượng bản sao khác đang tồn tại.

* 1. **Các đối tượng có trong trong prefab**

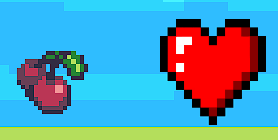
1. **Nhân vật phản diện:** Các đối nhân vật phản diệnđược tạo ra trong Scene sau đó thêm vào Prefab lúc này chúng trở thành những đối tượng bản sao của đối tượng gốc.

****

1. **Viên đạn:** Phương thức tấn công của người chơi và một số quái vật. Những viên đạn thường được các đối tượng khác gọi lên để xuất hiện.

****

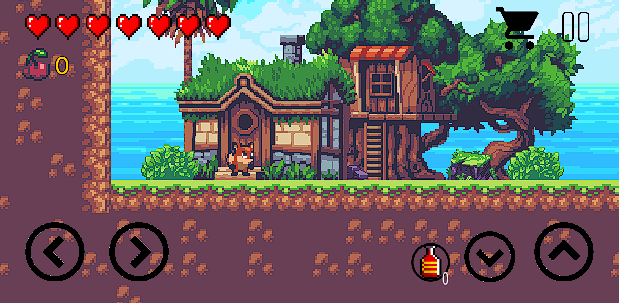
1. **Các vật phẩm:** Các đối tượng vật phẩm rớt ra khi quái vật bị đánh bại. Các đối tượng vật phẩm này thực hiện các chức năng khác nhau và biến mất khi Player chạm vào chúng.



1. **Giao diện điều khiển (UI)**
   1. **Thiết kế giao diện**

Giao diện điều khiển trên điện thoại là một phần quan trọng trong các trò chơi di động, nó giúp cho người chơi tương tác và điều khiển nhân vật của họ trên màn hình điện thoại thông qua các nút bấm. Các nút bấm trên giao diện điều khiển được thiết kế để thực hiện các hành động cơ bản, chẳng hạn như di chuyển, tấn công, nhảy... Các nút này có thể được đặt ở các vị trí khác nhau trên màn hình để tối ưu hóa việc sử dụng và tránh gây khó chịu cho người chơi. Ngoài các nút điều khiển cơ bản, trên giao diện điều khiển bao gồm các tính năng và chức năng khác như thanh máu, tiền, nút dừng game,...

Các nút điều khiển nhân vật sử dụng phương thức điều khiển trong script “Player Controller” của đối tượng Player và nút tấn công của nhân vật trong script “Player Combat”.



* 1. **Giao diện khi dừng game (Pause Menu)**

Pause Menu không được hiển thị khi trò chơi bắt đầu mà nó chỉ hiển thị lên khi người chơi nhấn vào nút dừng hoặc nếu như trò chơi bị tác động bởi các yếu tố bên ngoài như có điện thoại gọi đến hoặc game bị đẩy chạy xuống nền,...

Pause Menu là một lớp giao diện nằm bên trên giao diện điều khiển nhân vật vì vậy người chơi không thể tác động đến giao diện điều khiển nếu Pause Menu còn hiển thị.



Khi Pause Menu được gọi khi người chơi nhấn nút dừng hoặc game bị ẩn đi thì nó sẽ chỉnh thời gian trong game thời gian trong game bằng 0 sử dụng lệnh “Time.timeScale = 0f”, tức là trò chơi sẽ bị tạm dừng và không thể thực hiện được đến khi thời gian được khôi phục.

Khi người chơi nhấn vào nút “CONTINUE” để tiếp tục game hay “MAIN MENU” để trở về Menu chính thì thời gian trong game sẽ được chỉnh thành 1 khi đó game sẽ hoạt động lại như bình thường.

* 1. **Cửa hàng mua vật phẩm (Shop)**

Shop sẽ hiện ra khi người chơi nhấn vào nút xe hàng ở bên cạnh nút dừng. Cửa sổ shop sẽ hiện ra để mua vật phẩm.



Khi người chơi nhấn nút “MUA” nếu có đủ tiền thì tiền của Player sẽ bị trừ tương ứng và vật phẩm thuốc hồi máu sẽ được thêm vào “Player Balance”. Nếu không thì sẽ hiện thông báo không đủ tiền. Người chơi nhấn nút “THOÁT” sẽ ẩn giao diện Shop đi. Người chơi chỉ được mua tối đa 10 thuốc hồi máu.

Button sử dụng thuốc hồi máu sẽ được cập nhật số lượng mỗi khi thuốc hồi máu được mua hoặc sử dụng. Khi người chơi nhấn vào nút hồi máu nếu lượng máu hiện tại của Player nhỏ hơn lượng máu tối đa và số lượng thuốc hồi máu lớn hơn 0 thì người chơi mới được sử dụng thuốc hồi máu. Khi sử dụng thuốc hồi máu thành công, số lượng thuốc sẽ giảm đi 1 và lượng máu hiện tại của Player sẽ tăng thêm một.

Để tránh việc người chơi lạm dụng thuốc hồi máu sau mỗi lần hồi máu thành công, sẽ có khoảng thời gian hồi 10 giây. Trong lúc đó nút hồi máu không thể tương tác được và sau khi thời gian hồi kết thúc người chơi lại có thể tương tác với nút hồi máu.

1. **Âm thanh**
   1. **Khái niệm**

Là các âm thanh được phát ra trong trò chơi, bao gồm âm thanh của nhân vật, âm thanh của môi trường và các hiệu ứng âm thanh khác. Có hai loại âm thanh chính trong trò chơi là BGM và SFX.

* ***BGM (Background Music):*** Là âm thanh nền được phát trong suốt quá trình chơi game. BGM thường được thiết kế để tạo ra một không gian âm nhạc riêng biệt cho trò chơi, giúp tăng cảm giác hứng thú và tạo ra một bầu không khí đặc biệt cho người chơi. BGM thường có thời lượng dài và được lặp đi lặp lại trong suốt quá trình chơi game.
* ***SFX (Sound Effects):***Là các hiệu ứng âm thanh được sử dụng để tạo ra các âm thanh khác nhau trong trò chơi. SFX có thể là tiếng bắn đại, âm thanh lúc nhảy, tiếng quái vật kêu,... SFX thường được thiết kế để tương tác với hành động của người chơi hoặc các đối tượng trong trò chơi, giúp tăng tính tương tác và độ chân thực của trò chơi.
  1. **Thiết kế âm thanh**

Trong đối tượng “AudioManager” có script “Sound” bao gồm các thuộc tính:

* **name:** Tên của âm thanh.
* **clip:** Tệp âm thanh.
* **volume:** Âm lượng của âm thanh (từ 0 đến 1).
* **pitch:** Tốc độ phát của âm thanh (từ 0.1 đến 3).
* **SpatialBlend:** Tỷ lệ giữa âm thanh 2D và 3D (từ 0 đến 1).
* **Looping:** Cho phép âm thanh lặp lại hay không.

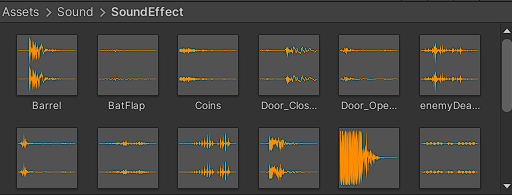
Các thông tin này được sử dụng để phát các âm thanh và điều chỉnh các thuộc tính của âm thanh. Phương thức sẽ tìm kiếm thông tin về âm thanh trong danh sách âm thanh do lập trình viên bỏ vào, và phát âm thanh. Nếu âm thanh được gọi không có tên trong danh sách thì sẽ in ra thông báo lỗi.

Quản lý âm thanh và các Button để điều chỉnh âm thanh trong giao diện Pause Menu của game, lưu trữ trạng thái âm thanh của BGM và SFX có đang bị tắt âm hay không. Khi người chơi nhấn nút âm thanh nếu âm thanh đang được phát, thì sẽ tắt âm thanh, cập nhật hình ảnh của button thành tắt âm. Ngược lại, nếu âm thanh đang bị tắt thì sẽ bật âm thanh và hình ảnh của button thành âm thanh đang bật.

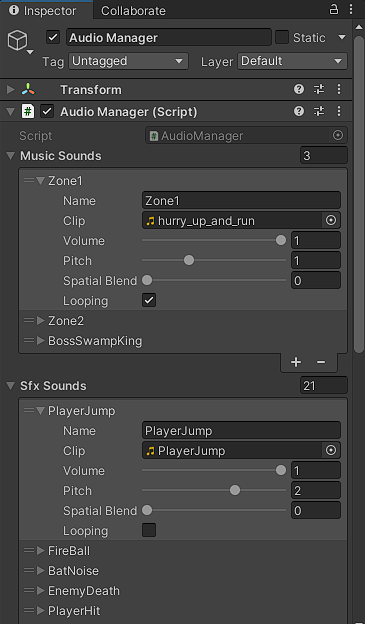


**Các bước để thêm âm thanh**

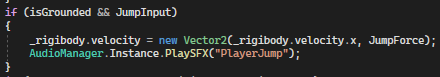
1. Thu thập các file âm thanh.



1. Kéo âm thanh vào Clip của BGM hoặc SFX sau đó đặt tên cho âm thanh và điều chỉnh các thông số khác nếu cần thiết.



1. Gọi âm thanh để sử dụng ví dụ như trong “Player Controller” khi Player thực hiện nhảy thì sẽ phát SFX có tên “PlayerJump”:



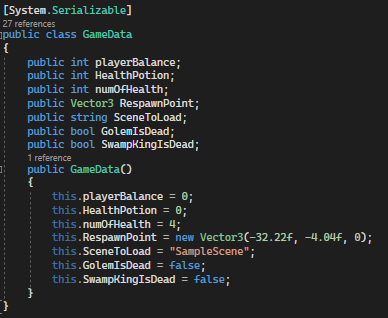
1. **Hệ thống lưu game**

Hệ thống lưu game dùng để quản lý dữ liệu trò chơi và lưu trữ thông tin vào một file JSON. Trong đó có ba script thực hiện chức năng chính để lưu thông tin: GameData, DataPersistenceManager và FileDataHandler.

**FileDataHandler:** Lưu trữ và tải dữ liệu game từ tệp tin. Nó có hai phương thức:

* Phương thức “Load” để tải và đọc dữ liệu từ tệp tin.
* Phương thức “Save” được sử dụng để lưu trữ dữ liệu vào tệp tin.

**GameData:**được sử dụng để lưu trữ thông tin về trạng thái của trò chơi, bao gồm số tiền của Player, số lượng bình thuốc hồi máu, số lượng máu tối đa, vị trí hồi sinh, tên Scene cần được tải và trạng thái bị đánh bại của hai con trùm trong trò chơi. Có phương thức khởi tạo mặc định để thiết lập giá trị ban đầu cho các thuộc tính.

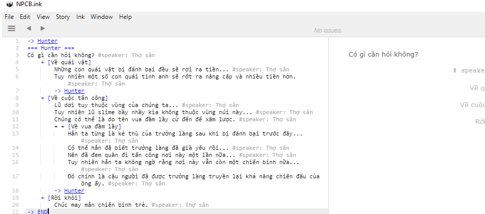


**DataPersistenceManager:** chịu trách nhiệm quản lý việc lưu trữ và tải dữ liệu trò chơi. Script này chứa ba phương thức chính để thực hiện một chức năng lưu dữ liệu game:

* Phương thức NewGame() thực hiện khi người chơi chọn “New Game” khởi tạo một file JSON mới với các giá trị ban đầu từ script “GameData” và thay thế file dữ liệu cũ nếu đã tồn tại.
* Phương thức LoadGame() để tải dữ liệu trò chơi từ tệp tin. Nếu không có dữ liệu được tìm thấy trong tệp tin thì nó sẽ không làm gì cả.
* Phương thức SaveGame() được dùng để ghi đè dữ liệu trò chơi vào file JSON.

1. **Đối thoại với các NPC**

Trên chuyến hành trình của mình Player sẽ bắt gặp các NPC, Player có thể nói chuyện được với họ để tiếp nhận thông tin và hỏi các câu hỏi liên quan. Để tạo một cuộc đối thoại với NPC trong Unity, sử dụng các công cụ như TextMeshPro và UI Canvas để hiển thị văn bản của NPC khi nói chuyện. Sử dụng Ink để viết câu chuyện cho NPC, khi tạo một file Ink thì nó cũng tự tạo một file JSON tương ứng để sử dụng làm một cuộc đối thoại trong game.

’

Đầu tiên, cần tạo một đối tượng NPC trong trò chơi, sử dụng Collider 2D và đặt thuộc tính “Is Trigger” của Collider thành True, Collider sẽ không gây ra va chạm nhưng sẽ kích hoạt sự kiện, khi đó Collider trở thành khu vực tương tác của NPC khi va chạm với Player. Thêm các UI Canvas, sử dụng TextMeshPro để hiển thị văn bản của cuộc đối thoại. Khi Player vào khu vực tương tác thì sẽ hiển thị một nút để người chơi tương tác với NPC, khi người chơi nhấn nút sẽ hiển thị hộp thoại của NPC mà người chơi đang tương tác.



Trong UI có các nút để cho người chơi lựa chọn các câu hỏi khác nhau. Khi người chơi chọn một câu hỏi, các luồng đối thoại trong file Ink của NPC để xác định câu trả lời tiếp theo và hiển thị nó trên hộp thoại.



**Script** “**DialogueManager” quản lý** **hiển thị và điều khiển hộp thoại của các NPC:**

Trong đó khai báo hộp thoại, khu vực để hiển thị nội dung hộp thoại, để hiển thị tên của người đang nói trong cuộc đối thoại, mảng để hiển thị các lựa chọn trong cuộc đối thoại nếu có, nội dung của cuộc đối thoại của NPC và một biến để kiểm tra xem có cuộc hội thoại nào đang diễn ra hay không. Trong script có các phương thức chính:

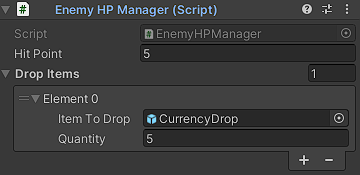
* Phương thức EnterDialogueMode() được sử dụng để bắt đầu một cuộc hội thoại mới. Trong phương thức này, script tạo một đối tượng Story mới từ file JSON chứa nội dung câu chuyện được bỏ vào cho NPC.
* Phương thức ContinueStory() được sử dụng để hiển thị nội dung tiếp theo của câu chuyện. Nếu câu chuyện vẫn chưa kết thúc. Nếu câu chuyện đã hết thì kết thúc cuộc hội thoại và đóng hộp thoại.
* Phương thức DisplayChoices() được sử dụng để hiển thị các lựa chọn cho người chơi. Script lấy ra danh sách các lựa chọn hiện tại từ đối tượng Story và hiển thị chúng lên giao diện hiển thị.
* Phương thức MakeChoice() xử lý lựa chọn của người chơi. Khi người chơi chọn một lựa chọn, script gọi phương thức ChooseChoiceIndex() để xác định lựa chọn được chọn và tiếp tục câu trả lời tương ứng.

1. **Nhân vật phản diện**

Trong trò chơi các nhân vật phản diện chính là các kẻ địch được thiết kế để tạo ra thử thách và khó khăn cho người chơi. Các kẻ địch trong trò chơi có thể có nhiều hình dạng, kích cỡ, sức mạnh và tính năng khác nhau. Những kẻ địch này đều được đặt cho Tag là “Enemy” để tương tác với Player và Layer được đặt là “Enemy” để chúng di chuyển mà không va chạm lẫn nhau.

* **Các Component được sử dụng chung:**
* Sprite Renderer:Hiển thị hình ảnh của đối tượng.
* Rigibody 2D:Mô phỏng vật lý cho đối tượng.
* Animator:Hoạt ảnh chuyển động của đối tượng.
* Box Collider 2D:Vùng va chạm của đối tượng.
* **Những script được dùng chung cho nhân vật phản diện:**
* **Enemy HP Manager:**

Quản lý lượng máu của đối tượng và một mảng các vật phẩm trong Prefab chứa thông tin về các vật phẩm và số lượng vật phẩm. Khi đối tượng bị tấn công, lượng máu sẽ giảm đi một và nếu lượng máu hiện tại nhỏ hơn hoặc bằng 0, thì sẽ thực hiện rớt các vật phẩm mà đối đượng có và hủy đối tượng đó.



* **Blink:**

Khi đối tượng bị Player đánh trúng. Trong một khoảnh khắc, màu sắc của sprite sẽ được thay đổi thành màu đỏ, sau đó trở lại màu trắng ban đầu sau 0.1s. Khi kết thúc đối tượng sẽ trở lại màu sắc ban đầu của nó.



* **Enemy Knock Back:**

Khi đối tượng này va chạm với một Collider2D khác. Nếu Collider2D đó có tag là "Player", thì sẽ thực hiện đẩy lùi Player

* 1. **Slime (Chất nhầy)**

Lượng máu: 5.

Rớt vật phẩm: 5 tiền.

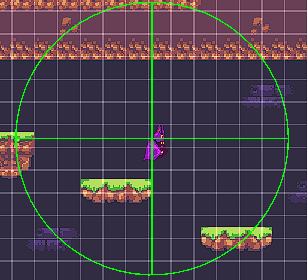
Cách tấn công: Di chuyển trái và phải

* **Script được sử dụng để xây dựng Slime:**

**Enemy AI Slime:**Được sử dụng để điều khiển hành vi di chuyển của Slime.

Các biến được khai báo gồm các đối tượng kiểm tra va chạm với mặt đất, và với tường, xác định hướng di chuyển hiện tại của Slime và tốc độ di chuyển của Slime.

Slime có hai đối tượng dùng để kiểm tra liên tục xem Slime có đang di chuyển trên mặt đất hay chạm tường. Slime sẽ đổi chiều di chuyển của nó nếu chạm trúng tường hoặc không chạm nền đất.

* 1. **Bat (Dơi)**

Lượng máu: 3

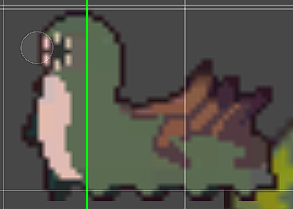
Rớt vật phẩm: 4 tiền

Cách tấn công: Bay đến vị trí của Player

* **Script được sử dụng để xây dựng Bat:**

**Enemy AI Bat:**Điềukhiển hành vi và di chuyển của Bat.

Các biến được khai báo gồm tốc độ di chuyển và khoảng cách tối đa mà Bat có thể nhìn thấy được Player. Nếu khoảng cách giữa Player và nó nhỏ hơn khoảng cách tối đa mà Bat thấy được Player hoặc nếu nó bị Player đánh trúng thì nó sẽ bay về phía Player.

* 1. **Worm (Sâu)**

Lượng máu: 6

Rớt vật phẩm: 10 tiền

Cách tấn công: Di chuyển đến vị trí của Player khi bị Player đánh trúng hoặc vào khu vực nó có thể thấy được. Nếu trong khu vực tấn công thì sẽ thực hiện phun đạn đến vị trí của Player.

* **Script được sử dụng để xây dựng Worm:**

**Worm:**Được sử dụng để điều khiển hành vi của Worm.

Các biến được khai báo để xác định tốc độ di chuyển và khoảng cách tối đa mà nó có thể nhìn thấy Player. Nếu Player trong khu vực mà Worm có thể thấy được nó sẽ bắt đầu tấn công Player. Worm sẽ phun đạn mỗi 4s nếu người chơi còn trong khu vực tấn công. Nếu Player nằm ngoài phạm vi tấn công thì nó sẽ bò đến vị trí của Player. Đồng thời xoay sprite của Worm cho phù hợp với vị trí của Player.

* 1. **Frog (Ếch)**

****Lượng máu: 10

Rớt vật phẩm: 15 tiền

Cách tấn công: Nhảy đến vị trí Player khi bị đánh trúng hoặc vào khu vực tấn công của nó

* **Script được sử dụng để xây dựng Frog:**

**Enemy AI Frog:**Được sử dụng để điều khiển hành vi của Frog.

Các biến được khai báo để xác định tốc độ di chuyển của Frog, kiểm tra nó đang chạm đất, và hướng di chuyển của nó, độ cao của Frog khi nhảy, khoảng cách tối đa mà Frog có thể nhìn thấy Player và thời gian mỗi lần nhảy của Frog để nó không nhảy liên tục. Nếu Player vào phạm vi tấn công của Frog hoặc tấn công nó thì Frog sẽ thực hiện nhảy đến vị trí của Player. Frog sau khi thực hiện xong bước nhảy sẽ kiểm tra nó đã chạm đất chưa và khoảng thời gian nghỉ mỗi lần nhảy đã kết thúc chưa thì nó mới thực hiện bước nhảy tiếp theo.

1. **Nhân vật phản diện là trùm (Boss)**

* ***Phòng Boss***

Tại một số màn chơi nhất định Player sẽ đến được phòng Boss. Khi Player chạm vào khu vực triệu hồi nó sẽ triệu hồi con boss mà nó cần gọi và bắt đầu đóng hai cửa để chặn đường lui của Player khi con Boss được triệu hồi thành công và mở cửa khi con trùm đó bị đánh bại hoặc người chơi bị đánh bại. Hệ thống sẽ đọc thông tin trong dữ liệu game nếu con boss mà nó cần triệu hồi đã bị đánh bại bởi người chơi thì sẽ không triệu hồi con boss đó nữa.



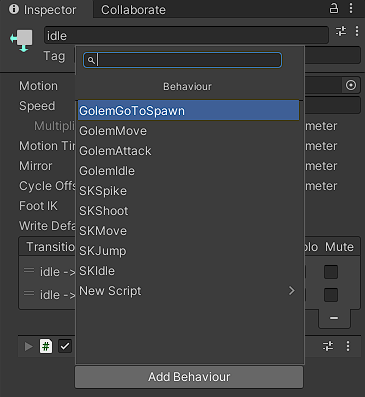
* ***Boss***

Boss là những kẻ địch có khả năng tấn công phức tạp hơn và có thể có nhiều điểm mạnh hơn so với các kẻ địch thông thường. Chúng được thiết kế để đánh bại bằng cách sử dụng các chiến thuật và kỹ năng của người chơi, và thường được xem là một thử thách lớn mà người chơi phải hạ gục trước khi chuyển qua khu vực kế tiếp.

* ***StateMachineBehaviour***

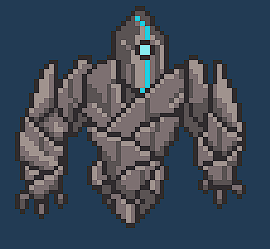
StateMachineBehaviour là một lớp trong Unity dùng để thêm hành vi đặc biệt vào các Animator của State Machine. Nó cho phép tùy chỉnh các Animator bằng cách thêm các hành vi riêng biệt vào Animator. Cơ chế hoạt động của nó là khi thêm một StateMachineBehaviour vào một Animator, nó sẽ được gắn kết với Animator đó và được gọi mỗi khi Animator đó được thực thi. Có thể sử dụng StateMachineBehaviour để thêm các hành vi đặc biệt vào Animator như thêm âm thanh, hiệu ứng hình ảnh, hoặc các hành động khác để tăng tính tương tác của trò chơi. StateMachineBehaviour cũng cho phép truy cập các thành phần khác trong trò chơi, chẳng hạn như các đối tượng của trò chơi, hoặc các thành phần khác để thực hiện các hành động phức tạp hơn.

Để có thể sử dụng StateMachineBehaviour cho Animator đầu tiên phải chọn Animator cần thêm script và chọn “Add Behaviour” sau đó chọn script đã có sẵn hoặc tạo một script mới cho Animator và đặt tên cho nó, những script này có lớp là StateMachineBehaviour thay vì MonoBehaviour như các script thông thường.



**Các phương thức của StateMachineBehaviour bao gồm:**

1. **OnStateEnter():** Phương thức này được gọi khi mới vào Animator. Nó cho phép thực hiện các hành động cần thiết để chuẩn bị cho trạng thái mới, giống với Start().
2. **OnStateUpdate():** Phương thức này được gọi mỗi khung hình khi Animator đang được thực thi. Nó cho phép thực hiện các chức năng cần thiết để duy trì trạng thái hiện tại, giống với Update().
3. **OnStateExit():** Phương thức này được gọi khi một Animator kết thúc và trước khi chuyển sang Animator mới. Nó cho phép thực hiện các chức năng để kết thúc Animator hiện tại.
4. **OnStateMove():** Phương thức này được gọi khi Animator chuyển từ Animator này sang Animator khác. Nó cho phép thực hiện các chức năng cần thiết để chuyển đổi giữa các Animator.
5. **OnStateMachineEnter():** Phương thức này được gọi khi State Machine bắt đầu chạy. Nó cho phép thực hiện các hành động cần thiết để chuẩn bị cho State Machine.
6. **OnStateMachineExit():** Phương thức này được gọi khi State Machine kết thúc. Nó cho phép thực hiện các hành động cần thiết để kết thúc State Machine.
   1. **Golem (Quái vật đá)**

****Lượng máu: 40

Rớt vật phẩm: 60 tiền, 1 vật phẩm tăng máu tối đa cho nhân vật chính, 1 Boss Golem Medal để chứng nhận đã đánh bại Boss Golem.

**Cách thức tấn công của Golem:**

* Di chuyển đến vị trí của Player.
* Tấn công Player bằng tia laze bắn từ tay của Golem.
* Tấn công bằng cách bắn một tia laze khổng lồ phía ở trên.

**Các Script được sử dụng để xây dựng Golem:**

1. **Boss Golem HP Manager:** Điều khiển lượng HP của Golem

Được sử dụng để quản lý lượng máu của đối tượng, hủy đối tượng, rớt vật phẩm nếu lượng máu về 0 và lưu thông tin vào dữ liệu game để xác nhận Boss đã bị đánh bại.

1. **Golem:**Điều khiển phương thức xoay Golem

Các biến được khai báo và khởi tạo ở đầu script gồm tốc độ, tọa độ Player, kiểm tra Golem đang tấn công, kiểm tra Golem có đang xoay sang phải hay không. Xoay Golem theo hướng phù hợp với Player.

1. **Golem Idle:** Golem về trạng thái mặc định không làm gì cả.

Khai báo timer để tính toán thời gian đổi sang Animator khác. Nếu timer bé hơn bằng 0 thì nó sẽ gọi trigger “Run” trong Animator thực hiện “GolemMove” để di chuyển Golem đến vị trí Player, nếu không thì thời gian sẽ tiếp tục giảm dần về 0.

1. **Golem Move:** Golem di chuyển đến vị trí nhân vật.

Golem di chuyển đến vị trí Player, nếu timer bé hơn 0 và khoảng cách giữa Player và Golem trong khoảng cách nhất định thì sẽ trigger “Attack” thực hiện “Golem Attack” để tấn công.

1. **Golem Attack:** Đợt tấn công bằng cách bắn tia laze của Golem

Golem sẽ phát âm thanh “GolemAttackBeam” và thực hiện animator tấn công của nó. Sau khi thực hiện xong đòn đánh thì nó sẽ trigger “LazeTrap” để thực hiện “Golem Go To Spawn”.



1. **Golem Go To Spawn:** Golem di chuyển đến “Spawn” và tấn công phía trên.

Golem di chuyển đến vị trí mà nó được triệu hồi đồng thời triệu hồi một tia laze khổng lồ từ Prefab đến vị trí chỉ định để bắn các bục ở trên để tránh việc Player đứng mãi trên các bục. Sau khi thực hiện xong nó sẽ trigger “Idle” để quay lại “Golem Idle”



* 1. **Swamp King (Vua đầm lầy)**

Lượng máu: 50

Rớt vật phẩm: 80 tiền, 1 vật phẩm tăng máu tối đa cho nhân vật chính, 1 Swamp King Medal để chứng nhận đã đánh bại Boss Swamp King.

**Cách thức tấn công của Swamp King:**

* Di chuyển đến vị trí của Player.
* Nhảy đến vị trí của Player.
* Bắn một viên đạn đến vị trí của Player.
* Thực hiện tấn công bằng gai khi Player đến quá gần

**Các Script được sử dụng để xây dựng Swamp King:**

1. **Swamp King HP Manager:**Điều khiển lượng HP của Swamp King

Được sử dụng để quản lý lượng máu của đối tượng, hủy đối tượng, rớt vật phẩm nếu lượng máu về 0 và lưu thông tin vào dữ liệu game để xác nhận Swamp King đã bị đánh bại.

1. **Swamp King:**Quản lý các hành động và đòn tấn công của Swamp King

Khai báo các biến như kiểm tra đối tượng đang tiếp xúc với mặt đất, độ cao khi nhảy, khu vực để bắn viên đạn khi tấn công, đối tượng viên đạn được bắn ra. Script này lưu trữ các phương thức để StateMachineBehaviour sử dụng vào các đòn tấn công của đối tượng: Phương thức bắn đạn đến vị trí Player và phương thức nhảy đến vị trí của Player.

1. **SKIdle:** Đối tượng không làm gì cả, thực hiện hoạt ảnh nhàn rỗi.

Khai báo timer và thực hiện đếm ngược nếu timer bé hơn bằng 0 thì sẽ trigger “Move” để thực hiện “SKMove”

1. **SKMove:**Đối tượng di chuyển đến nhân vật và thực hiện đòn tấn công khác nhau.

Khai báo timer và rand để thay đổi đòn đánh. Swamp King thực hiện di chuyển đến vị trí Player. Nếu timer bé hơn bằng 0 thì sẽ thực hiện tấn công, nếu khoảng cách giữa Player và đối tượng bé hơn khoảng cách nhất định thì sẽ trigger đòn đánh “Spike”. Nếu không thì kiểm tra nếu rand bằng 0 thì trigger đòn đánh “Jump” sau đó gán rand bằng 1. Nếu rand bằng 1 thì sẽ trigger đòn đánh “Shoot” và gán rand bằng 0 điều này sẽ làm luân phiên giữa hai đòn đánh.

1. **SKSpike:**Đòn tấn công bằng gai.

NếuPlayer đến quá gần khoảng cách với Swamp King thì nó sẽthực hiện hoạt ảnh tấn công bằng gai từ bên dưới sau đó khi thực hiện xong đòn tấn công thì sẽ quay lại “SKMove”.



1. **SKJump:** Đòn tấn công nhảy.

Gọi phương thức nhảy tại script “SwampKing” để thực hiện nhảy đến vị trí hiện tại của Player. Sau đó nếu nếu đối tượng tiếp đất thì quay lại “SKIdle”.



1. **SKShoot:**Đòntấn công bắn đạn.

Bắn một viên đạn đến vị trí của Player và sau khi bắn xong sẽ quay lại “SKIdle”.



1. **KẾT LUẬN**
2. **Kết quả của việc làm game**

Em đã tiếp thu được rất nhiều kiến thức mới và tích lũy được một số kinh nghiệm quan trọng về việc phát triển game, hiểu rõ hơn về quá trình phát triển game từ việc thiết kế, lập trình, đến kiểm thử và cho ra sản phẩm. Học được cách sử dụng Unity để phát triển game, bao gồm cách tạo các đối tượng, tạo môi trường, xử lý sự kiện và tối ưu hóa hiệu suất để game chạy mượt hơn, các hệ thống đồ họa, âm thanh và các hệ thống hỗ trợ cho game của mình để tạo ra một sản phẩm hoàn chỉnh.

1. **Những gì đã đạt được**

Sau quá trình phát triển, em đã hoàn thành game với các mục tiêu đã đề ra ban đầu. Game có giao diện đơn giản và dễ sử dụng, hình ảnh đẹp mắt và âm thanh sống động. Các lỗi trong game đa số đã được sửa để không ảnh hưởng đến trải nghiệm người dùng. Nhân vật chính đã hoàn thiện những tính năng cơ bản như di chuyển, nhảy, tấn công và thu thập vật phẩm. Những nhân vật phản diện thực hiện chức năng như đã được lập trình. Em đã tạo ra một loạt các khu vực đa dạng và phong phú để người chơi có thể khám phá. Tóm lại, sản phẩm của em đã đạt được nhiều thành tựu đáng kể trong việc phát triển và em hy vọng rằng sản phẩm của mình sẽ mang lại trải nghiệm chơi game tốt nhất cho người chơi.

1. **Hạn chế**

Tilemap vẫn còn một lỗi là có hiện tượng những đường kẻ giữa các ô trong tilemap bị giật đứt, điều này xảy ra do không có sự đồng bộ giữa tốc độ vẽ khung hình và tốc độ cập nhật của tilemap. Một số chữ khi đối thoại với NPC có thể bị hỏng, điều này là do font chữ được sử dụng không hỗ trợ một số dấu câu. Có thể vẫn còn một vài lỗi tiềm ẩn trong game.

Trong quá trình phát triển game trong vòng 2 tháng, thách thức lớn nhất mà em gặp phải là về thời gian. Với hạn chế về thời gian như vậy, em cần phải tập trung hoàn thành những tính năng chính của game và giảm đi những tính năng phụ khác.

1. **Kết luận và hướng phát triển trong tương lai**

Tổng kết lại, em đã hoàn thành game với các tính năng cơ bản và đạt được toàn bộ mục tiêu đã đề ra ban đầu. Tuy nhiên, để có phát triển game thêm phần đa dạng và tăng trải nghiệm cho người chơi, em cần phải tiếp tục học hỏi các tính năng mới và cải thiện các kiến thức hiện có để có thể phát triển game tốt hơn trong tương lai.

**TÀI LIỆU KHAM THẢO**

[1] Tài liệu tiếng anh của Unity, <https://docs.unity.com/>

[2] Tài liệu về lập trình hướng đối tượng trên elearning.

[3] Cộng đồng Unity, <https://community.unity.com/>

[4] Tài liệu học Unity trên Coursera, <https://www.coursera.org/courses?query=unity>

[5] Các khóa hướng dẫn của Trever Mock, <https://www.youtube.com/@TreverMock>

**TÀI NGUYÊN GAME**

[1] Asset Sunny Land, <https://assetstore.unity.com/packages/2d/characters/sunny-land103349>

[2] Ink Unity Integration, <https://assetstore.unity.com/packages/tools/integration/ink-unity-integration-60055>

[3] Các âm thanh miễn phí tại Free Sound, <https://freesound.org/>

**TẢI GAME**

Link Mega:

<https://mega.nz/file/llYmnZAa#_5o5_0oVB5-nXDSC8blZwpt5SOk8OFBIlV8BHPrTT1o>

**PHỤ LỤC**

**Các Tag và Layer của các đối tượng trong game:**

|  |  |  |
| --- | --- | --- |
| **Đối tượng** | **Tag** | **Layer** |
| **Nhân vật chính** | Player | Player |
| **Bản đồ, nền đất** | Ground | Ground |
| **Nhân vật phản diện** | Enemy | Enemy |
| **Viên đạn của nhân vật chính** | Projectile |  |
| **Viên đạn của kẻ thù** | EnemyBullet |  |

**Script thực hiện các chức năng chính của Player**

* 1. **Phương thức di chuyển của Player**

Người chơi nhấn nút di chuyển hướng nào thì Player di chuyển sang hướng đó. Người chơi thả tay khỏi nút di chuyển thì Player dừng di chuyển.

Player chạm đất thì “isGrounded” là true và không chạm thì là false. Người chơi nhấn nút nhảy khi “isGrounded” là true thì Player thực hiện nhảy. Người chơi nhấn nút nhảy “isGrounded” là false thì Player sẽ không thực hiện nhảy.

private void FixedUpdate()

{

if (canMove && !DialogueManager.GetInstance().dialoguePlaying)

{

UpdateAnimationState();

if (isMovingLeft && isMovingRight == false)

{

transform.Translate(MovementSpeed \* Time.deltaTime, 0, 0);

transform.rotation = Quaternion.Euler(0, 180, 0);

}

else if (isMovingRight && isMovingLeft == false)

{

transform.Translate(MovementSpeed \* Time.deltaTime, 0, 0);

transform.rotation = Quaternion.Euler(0, 0, 0);

}

else

{

transform.Translate(0 \* Time.deltaTime, 0, 0);

}

if (isGrounded && JumpInput)

{

\_rigibody.velocity = new Vector2(\_rigibody.velocity.x, JumpForce);

AudioManager.Instance.PlaySFX("PlayerJump");

}

if (JumpInput == false && \_rigibody.velocity.y > 0)

{

\_rigibody.velocity = new Vector2(\_rigibody.velocity.x, \_rigibody.velocity.y / JumpRelease);

}

}

}

* 1. **Tấn công**

Người chơi nhấn nút tấn công thì Player thực hiện tấn công. Người chơi thả nút tấn công thì thì Player dừng tấn công.

private void Update()

{

if (isShooting == true && cooldownTimer > attackCooldown && !DialogueManager.GetInstance().dialoguePlaying)

{

Attack();

}

cooldownTimer += Time.deltaTime;

}

public void isAttack()

{

isShooting = true;

}

public void isnotAttack()

{

isShooting = false;

}

private void Attack()

{

cooldownTimer = 0;

Instantiate(projectilePrefab, firePoint.position, transform.rotation);

}

* 1. **Điều khiển máu của Player, nhận sát thương và hồi sinh**

Lượng máu của Player có điều kiện không được phép lớn hơn lượng máu tối đa. Nếu Player nhận sát thương thì máu sẽ bị trừ đi một và bất tử trong thời gian ngắn. Nếu lượng máu hiện tại của Player bé hơn bằng 0 thì nó sẽ thực hiện hồi sinh, nếu hồi sinh thì lượng máu của Player sẽ hồi bằng lượng máu tối đa.

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.tag == "DeathZone")

{

Respawn();

DataPersistenceManager.instance.SaveGame();

}

if (collision.gameObject.tag == "Enemy")

{

AudioManager.Instance.PlaySFX("PlayerHit");

TakeDamage();

}

if(collision.gameObject.tag == "heart")

{

numOfHealth += 1;

health = numOfHealth;

DataPersistenceManager.instance.SaveGame();

}

}

private void FixedUpdate()

{

if (health > numOfHealth)

{

health = numOfHealth;

}

for (int i= 0; i< hearts.Length; i++)

{

if(i < health)

{

hearts[i].sprite = heart;

}

else

{

hearts[i].sprite = emtpy\_heart;

}

if(i < numOfHealth)

{

hearts[i].enabled = true;

} else

{

hearts[i].enabled = false;

}

if(health <= 0)

{

Respawn();

}

}

}

public void TakeDamage()

{

health = health - 1;

if(health > 0)

{

StartCoroutine(Invunerability());

}

}

private IEnumerator Invunerability()

{

Physics2D.IgnoreLayerCollision(9, 8, true);

for(int i = 0; i < numberFlash; i++)

{

\_sprite.color = new Color(1, 0, 0, 0.5f);

yield return new WaitForSeconds(invincibleDuration / (numberFlash \* 2));

\_sprite.color = Color.white;

yield return new WaitForSeconds(invincibleDuration / (numberFlash \* 2));

}

Physics2D.IgnoreLayerCollision(9, 8, false);

}

private void Respawn()

{

if (SceneToLoad == "")

{

SceneManager.LoadScene("SampleScene");

}

SceneManager.LoadScene(SceneToLoad);

transform.position = respawnPoint;

health = numOfHealth;

}

1. **Script của các cánh cổng để Player di chuyển giữa các Scene.**
   1. **Di chuyển đến Scene khác.**

Lưu exitName vào lastExitName của PlayerPrefs khi Player chạm vào cổng.

public string sceneToLoad;

public string exitName;

private void OnCollisionEnter2D(Collision2D collision)

{

if(collision.gameObject.tag == "Player")

{

PlayerPrefs.SetString("lastExitName", exitName);

SceneManager.LoadScene(sceneToLoad);

}

}

* 1. **Lối ra sau khi di chuyển.**

Đọc từ lastExitName từ PlayerPrefs để dịch chuyển Player.

public string lastExitName;

void Start()

{

if (PlayerPrefs.GetString("lastExitName") == lastExitName)

{

PlayerManager.Instance.transform.position = transform.position;

}

}

1. **Script âm thanh.**
   1. **Script chứa các thông tin để điều chỉnh âm thanh.**

public class Sound

{

public string name;

public AudioClip clip;

[Range(0f, 1f)]

public float volume;

[Range(.1f, 3f)]

public float pitch;

[Range(0f, 1f)]

public float SpatialBlend;

public bool Looping;

}

* 1. **Phương thức phát âm thanh**

Nếu tên âm thanh được yêu cầu phát không có trong danh sách âm thanh thì hiển thị lỗi không thì phát âm thanh theo các thông tin được điều chỉnh.

public void PlayMusic(string name)

{

Sound s = Array.Find(musicSounds, x => x.name == name);

if (s == null)

{

Debug.Log("Sound not found");

}

else

{

musicSource.clip = s.clip;

musicSource.volume = s.volume;

musicSource.pitch = s.pitch;

musicSource.loop = s.Looping;

musicSource.Play();

* 1. **Phương thức bật và tắt âm thanh**

Khi nhấn nút nếu âm thanh đang bật thì tắt âm và ngược lại.

public void BGMpress()

{

if (BGMmuted == false)

{

BGMmuted = true;

musicSource.mute = true;

}

else

{

BGMmuted = false;

musicSource.mute = false;

}

Save();

UpdateButtonIcon();

}

public void SFXpress()

{

if (SFXmuted == false)

{

SFXmuted = true;

sfxSource.mute = true;

}

else

{

SFXmuted = false;

sfxSource.mute = false;

}

Save();

UpdateButtonIcon();

}

hiển thị nút tắt âm khi âm thanh đang bật và hiển thị nút bật âm thanh khi âm thanh đang tắt.

private void UpdateButtonIcon()

{

if (BGMmuted == false)

{

BGMButton.enabled = true;

BGMMuteButton.enabled = false;

}

else

{

BGMButton.enabled = false;

BGMMuteButton.enabled = true;

}

if (SFXmuted == false)

{

SFXButton.enabled = true;

SFXMuteButton.enabled = false;

}

else

{

SFXButton.enabled = false;

SFXMuteButton.enabled = true;

}

}

1. **Script của hệ thống lưu game**
   1. **Tạo file JSON mới khi nhấn New Game**

public void NewGame()

{

this.gameData = new GameData();

}

* 1. **Đọc dữ liệu game từ tệp khi Load Game**

Nếu không có dữ liệu game thì không thực hiện gì và nếu có dữ game thì bắt đầu đọc dữ liệu trong tệp tin và tiếp tục game theo thông tin trong đó.

public void LoadGame()

{

this.gameData = dataHandler.Load();

if(this.gameData == null)

{

Debug.Log("No save file was found");

return;

}

foreach(IDataPersistence dataPersistenceObj in dataPersistenceObjects)

{

dataPersistenceObj.LoadData(gameData);

}

}

* 1. **Lưu game vào tệp**

Lưu thông tin về dữ liệu game hiện tại vào tệp tin.

public void SaveGame()

{

foreach(IDataPersistence dataPersistenceObj in dataPersistenceObjects)

{

dataPersistenceObj.SaveData(gameData);

}

dataHandler.Save(gameData);

}

1. **Script đối thoại với NPC**
   1. **Phương thức khi bắt đầu và kết thúc cuộc đối thoại**

Khi bắt đầu cuộc đối thoại cuộc đối thoại hiện tại sẽ lấy từ file JSON Ink đưa vào trong NPC và hiển thị hộp thoại.

public void EnterDialogueMode(TextAsset inkJSON)

{

currentStory = new Story(inkJSON.text);

dialoguePlaying = true;

dialoguePannel.SetActive(true);

ContinueStory();

}

Khi tắt hộp thoại khi cuộc đối thoại kết thúc.

private void ExitDialogueMode()

{

dialoguePlaying= false;

dialoguePannel.SetActive(false);

dialogueText.text = "";

}

* 1. **Phương thức tiếp tục cuộc đối thoại**

Nếu cuộc đối thoại vẫn có thể tiếp tục thì sẽ tiếp tục câu đối thoại tiếp theo và hiển thị tên nhân vật và lựa chọn nếu có. Nếu không tiếp tục được nữa thì kết thúc.

public void ContinueStory()

{

if (currentStory.canContinue)

{

dialogueText.text = currentStory.Continue();

DisplayChoices();

HandleTags(currentStory.currentTags);

}

else

{

ExitDialogueMode();

}

}

* 1. **Hiển thị và lựa chọn các câu hỏi**

Nếu số lượng lựa chọn lớn hơn số lượng nút cho phép hiển thị tối đa thì in ra lỗi. Hiển thị mỗi lựa chọn lên màn hình.

private void DisplayChoices()

{

List<Choice> currentChoices = currentStory.currentChoices;

if(currentChoices.Count > choices.Length)

{

Debug.Log("More choice given than UI can support ");

}

int index = 0;

foreach(Choice choice in currentChoices)

{

choices[index].gameObject.SetActive(true);

choicesText[index].text = choice.text;

index++;

}

for(int i = index; i < choices.Length; i++)

{

choices[i].gameObject.SetActive(false);

}

}

Với mỗi lựa chọn được người chơi chọn lựa thì sẽ cho ra câu trả lời tương ứng trong cuộc đối thoại và tiếp tục câu chuyện.

public void MakeChoice(int choiceIndex)

{

currentStory.ChooseChoiceIndex(choiceIndex);

dialogueText.text = currentStory.Continue();

}

1. **Script quản lý giao diện dừng game.**
   1. **Dừng game.**

public class PauseButton : MonoBehaviour

{

public GameObject PauseMenu;

public void ShowMenu()

{

PauseMenu.SetActive(true);

Time.timeScale = 0f;

}

}

* 1. **Tiếp tục game và thoát ra menu chính.**

public class PauseConfig : MonoBehaviour

{

public GameObject PauseMenu;

public void Continue()

{

PauseMenu.SetActive(false);

Time.timeScale = 1.0f;

}

public void Exit()

{

Time.timeScale = 1.0f;

SceneManager.LoadScene("MainMenu");

}

}

1. **Script chung quản lý nhân vật phản diện.**
   1. **Enemy HP Manager**

public class EnemyHPManager : MonoBehaviour

{

public int HitPoint;

public DropManager[] dropItems;

public void TakeDamage(int damage)

{

HitPoint -= damage;

if (HitPoint <= 0)

{

DropItem();

AudioManager.Instance.PlaySFX("EnemyDeath");

Destroy(gameObject);

}

}

private void DropItem()

{

foreach (DropManager dropManager in dropItems)

{

for (int i = 0; i < dropManager.quantity; i++)

{

Instantiate(dropManager.itemToDrop, transform.position, Quaternion.identity);

}

}

}

* 1. **Blink**

public class Blink : MonoBehaviour

{

public SpriteRenderer sprite;

public void OnTriggerEnter2D(Collider2D collision)

{

if(collision.gameObject.tag == "Projectile")

{

StartCoroutine(Flash());

}

}

public IEnumerator Flash()

{

sprite.color = Color.red;

yield return new WaitForSeconds(0.1f);

sprite.color = Color.white;

}

}

* 1. **Enemy Knock Back**

public class EnemyKnockBack : MonoBehaviour

{

private void OnCollisionEnter2D(Collision2D collision)

{

if (collision.gameObject.tag == "Player")

{

collision.gameObject.GetComponent<Health>().KnockBack(collision.GetContact(0).normal);

}

}

}

1. **Script điều khiển hành vi của những nhân vật phản diện**
   1. **Slime**

private void FixedUpdate()

{

\_rigibody.velocity= Vector2.right \* MovementSpeed \* Time.deltaTime;

isGrounded = Physics2D.OverlapCircle(GroundCheck.transform.position, circleRadius, GroundLayer);

isWall = Physics2D.OverlapCircle(WallCheck.transform.position, circleRadius, GroundLayer);

if (!isGrounded)

{

Flip();

}

else if (isWall)

{

Flip();

}

}

private void Flip()

{

transform.Rotate(new Vector3(0, 180, 0));

}

* 1. **Bat**

void Update()

{

float distance = Vector2.Distance(Player.position, transform.position);

if (distance < lineOfSite)

{

Engage = true;

}

if (Engage || isHit)

{

FlyToward();

}

}

private void FlyToward()

{

transform.position = Vector2.MoveTowards(this.transform.position, Player.position, speed \* Time.deltaTime);

if (Player.position.x < transform.position.x)

{

transform.rotation = Quaternion.Euler(0, 180, 0);

}

else

{

transform.rotation = Quaternion.Euler(0, 0, 0);

}

}

* 1. **Worm**

void Update()

{

float distance = Vector2.Distance(player.position, transform.position);

if (distance < lineOfSite)

{

Engage = true;

}

if (Engage || isHit)

{

if (Vector2.Distance(player.position, rb.position) < 7f)

{

if (timer > 4)

{

Spit();

timer = 0;

}

else

{

timer += Time.deltaTime;

}

}

else if(Vector2.Distance(player.position, rb.position) >= 5f)

{

GoToward();

animator.SetTrigger("walk");

}

}

}

private void Spit()

{

Instantiate(SpitBall, attackPoint.position,transform.rotation);

AudioManager.Instance.PlaySFX("Spit");

}

* 1. **Frog**

private void FixedUpdate()

{

isGrounded = Physics2D.OverlapBox(groundJump.position, boxSize, 0,groundLayer);

float distance = Vector2.Distance(player.position, transform.position);

if (distance < lineOfSite)

{

Engage = true;

}

if (Engage || isHit)

{

if (isGrounded)

{

if (timer > 2)

{

jump();

timer = 0;

}

else

{

timer += Time.deltaTime;

}

}

}

}

private void jump()

{

float distanceFromPlayer = player.position.x - transform.position.x;

AudioManager.Instance.PlaySFX("Frog");

if (distanceFromPlayer > 10f)

{

distanceFromPlayer = 10f;

}

else if (distanceFromPlayer < -10f)

{

distanceFromPlayer = -10f;

}

rb.AddForce(new Vector2(distanceFromPlayer, jumpHeight), ForceMode2D.Impulse);

}

* 1. **Các Script của Golem**
     1. **Golem Idle**

timer = Random.Range(minTime, maxTime);

public override void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

if (timer <= 0)

{

animator.SetTrigger("Run");

}

else

{

timer -= Time.deltaTime;

}

}

* + 1. **Golem Move**

public override void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

Vector2 target = new Vector2(player.position.x, rb.position.y);

Vector2 newPos = Vector2.MoveTowards(rb.position, target, speed \* Time.fixedDeltaTime);

rb.MovePosition(newPos);

if (timer <= 0)

{

if (Vector2.Distance(player.position, rb.position) <= attackRange)

{

animator.SetTrigger("Attack");

}

}

else

{

timer -= Time.deltaTime;

}

}

* + 1. **Golem Attack**

public override void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

if (timer <= 0)

{

animator.SetTrigger("LazeTrap");

}

else

{

timer -= Time.deltaTime;

}

}

* + 1. **Golem Go To Spawn**

public override void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

Vector2 target = new Vector2(Spawn.position.x, rb.position.y);

Vector2 newPos = Vector2.MoveTowards(rb.position, target, speed \* Time.fixedDeltaTime);

if (!hasInstantiated)

{

Instantiate(laze, LazePoint.position, Quaternion.identity);

AudioManager.Instance.PlaySFX("GolemAttackBeam");

hasInstantiated = true;

}

rb.MovePosition(newPos);

if (timer <= 0)

{

animator.SetTrigger("Idle");

}

else

{

timer -= Time.deltaTime;

}

}

* 1. **Các Script của Swamp King**
     1. **SK Idle**

public override void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

if (timer <= 0)

{

animator.SetTrigger("Move");

}

else

{

timer -= Time.deltaTime;

}

}

* + 1. **SK Move**

public override void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

Vector2 target = new Vector2(player.position.x, rb.position.y);

Vector2 newPos = Vector2.MoveTowards(rb.position, target, speed \* Time.fixedDeltaTime);

rb.MovePosition(newPos);

if (timer <= 0)

{

if (Vector2.Distance(player.position, rb.position) < 3.8f)

{

animator.SetTrigger("Spike");

}

else

{

if(rand == 0)

{

AudioManager.Instance.PlaySFX("SKprep");

animator.SetTrigger("Jump");

rand = 1;

}

else if(rand == 1)

{

animator.SetTrigger("Shoot");

rand = 0;

}

}

}

else

{

timer -= Time.deltaTime;

}

}

* + 1. **SK Shoot**

public void SKShoot()

{

StartCoroutine(shoot());

}

IEnumerator shoot()

{

yield return new WaitForSeconds(0.5f);

AudioManager.Instance.PlaySFX("Spit");

Instantiate(SKbullet, SKShootPoint.position, transform.rotation);

}

public override void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

if (!hasInstantiated)

{

sk.SKShoot();

hasInstantiated = true;

}

if (timer <= 0)

{

animator.SetTrigger("Idle");

}

else

{

timer -= Time.deltaTime;

}

}

* + 1. **SK Spike**

public override void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

if (timer <= 0)

{

animator.SetTrigger("Move");

}

else

{

timer -= Time.deltaTime;

}

}

* + 1. **SK Jump**

public void jump()

{

float distanceFromPlayer = Player.position.x - transform.position.x;

AudioManager.Instance.PlaySFX("SKJump");

if (isGrounded)

{

rb.AddForce(new Vector2(distanceFromPlayer, jumpHeight), ForceMode2D.Impulse);

}

}

public override void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)

{

if(!Jump)

{

sk.jump();

Jump = true;

}

if (timer <= 0 && sk.isGrounded)

{

animator.SetTrigger("Idle");

}

else

{

timer -= Time.deltaTime;

}

}