# TRƯỜNG ĐẠI HỌC KINH TẾ QUỐC DÂN

**2024-2025.1 CNTT Nguyễn Anh Tuấn 11218488**

# TRƯỜNG CÔNG NGHỆ

--------🙞🙞🕮🙜🙜--------



**KHÓA LUẬN TỐT NGHIỆP**

**Xây dựng game chiến thuật theo lượt (Turn base RPG)**

**Tên sinh viên : Nguyễn Anh Tuấn**

**Giảng viên hướng dẫn : Phạm Minh Hoàn**

**Hà Nội, 11/2024**

**TRƯỜNG ĐẠI HỌC KINH TẾ QUỐC DÂN   
TRƯỜNG CÔNG NGHỆ**--------🙞🙞🕮🙜🙜--------



**KHÓA LUẬN TỐT NGHIỆP**

**Xây dựng game chiến thuật theo lượt (Turn base RPG)**

**Tên sinh viên : Nguyễn Anh Tuấn**

**Mã sinh viên : 11218488**

**Lớp : Công nghệ thông tin K63B**

**Ngành : Công nghệ thông tin**

**Khoa : Công nghệ thông tin**

**Giảng viên hướng dẫn : Phạm Minh Hoàn**

*(Chữ ký GVHD)*

**Hà Nội, 11/2024**

MỤC LỤC

[CHƯƠNG 1: LÝ THUYẾT 6](#_Toc183434897)

[1.1 Mục tiêu và phương pháp nghiên cứu 6](#_Toc183434898)

[1.2 Khái niệm về Game 6](#_Toc183434899)

[1.3 Khái niệm và yêu cầu game chiến thuật 11](#_Toc183434900)

[1.4 Quy trình triển khai Game chiến thuật? 13](#_Toc183434901)

[1.5 Lựa chọn công cụ phát triển 14](#_Toc183434902)

[1.6 Đối tượng tham gia hệ thống 18](#_Toc183434903)

[1.7 Lịch sử hình thành của Godot Engine 19](#_Toc183434904)

[1.8 Các thành phần trong Godot 20](#_Toc183434905)

[CHƯƠNG 2: ỨNG DỤNG XÂY DỰNG MÔ HÌNH TRÒ CHƠI 24](#_Toc183434906)

[2.1 Mô tả bài toán 24](#_Toc183434907)

[2.2 Nội dung của trò chơi 25](#_Toc183434908)

[2.3 Đầu vào của hệ thống game 28](#_Toc183434909)

[2.3.1 Đầu vào từ người chơi 28](#_Toc183434910)

[2.3.2 Đầu vào từ AI và hệ thống 28](#_Toc183434911)

[2.3.3 Đầu vào từ dữ liệu tài nguyên 28](#_Toc183434912)

[2.3.4 Đầu vào từ cấu trúc kịch bản và Cutscene 29](#_Toc183434913)

[CHƯƠNG 3: CÀI ĐẶT MÔI TRƯỜNG 30](#_Toc183434914)

[3.1 Xây dựng thế giới Overworld 30](#_Toc183434915)

[3.2 Xây dựng người chơi và Npc 32](#_Toc183434916)

[3.3 Xây dựng nhân vật di chuyển 33](#_Toc183434917)

[3.4 Xây dựng camera theo dõi nhân vật 34](#_Toc183434918)

[3.5 Xây dựng Popup 35](#_Toc183434919)

[3.6 Xây dựng hệ thống đối thoại (Dialog) 36](#_Toc183434920)

[3.7 Xây dựng Combat arena system 36](#_Toc183434921)

[3.8 Xây dụng lớp Battler 38](#_Toc183434922)

[3.9 Xây dựng Stats cho các Battler 43](#_Toc183434923)

[3.10 Xây dựng Animation cho các Battler 45](#_Toc183434924)

[3.11 Xây dựng Battler action 46](#_Toc183434925)

[3.12 Xây dụng Quest 48](#_Toc183434926)

[3.13 Xây dựng chức năng cài đặt (Setting) 49](#_Toc183434927)

[3.14 Xây dựng Interaction 51](#_Toc183434928)

[3.15 Xây dựng Trigger 53](#_Toc183434929)

[3.16 Xây dựng Equipment 54](#_Toc183434930)

[3.17 Xây dựng hệ thống Shop 58](#_Toc183434931)

[3.18 Xây dựng Cutscene 61](#_Toc183434932)

[3.19 Xây dựng hệ thống Inventory 62](#_Toc183434933)

[3.20 Xây dựng level 64](#_Toc183434934)

[3.21 Xây dựng chức năng Buff và Debuff 65](#_Toc183434935)

[CHƯƠNG 4: ĐÁNH GIÁ KẾT QUẢ 68](#_Toc183434936)

LỜI CAM ĐOAN

Tôi xin cam đoan rằng đề tài “**Xây dựng game chiến thuật theo lượt (Turn base RPG)**” được tiến hành một cách minh bạch, công khai. Toàn bộ nội dung và kết quả được dựa trên sự cố gắng cũng như sự nỗ lực của bản thân cùng với sự giúp đỡ không nhỏ từ thầy cô hướng dẫn.

Tôi xin cam đoan kết quả nghiên cứu được đưa ra trong đồ án là trung thực và không sao chép hay sử dụng kết quả của bất kỳ đề tài nghiên cứu nào tương tự.

Tôi sẵn sàng chịu toàn bộ trách nhiệm nếu phát hiện rằng có bất kỳ sự sao chép kết quả nghiên cứu nào trong đồ án này.

*Hà Nội, ngày 25 tháng 11 năm 2024*

**Sinh viên**

**(ký, ghi rõ họ tên)**

LỜI CẢM ƠN

Trong bối cảnh ngành công nghiệp game ngày càng phát triển, việc tạo ra những trò chơi độc đáo và hấp dẫn là nhu cầu thiết yếu của các nhà phát triển. Một trong những thể loại được yêu thích nhất hiện nay chính là trò chơi RPG theo lượt. Với sự ra mắt của Godot 4, một công cụ lập trình mã nguồn mở, việc xây dựng một game RPG turn base trở nên dễ dàng hơn bao giờ hết. Bài báo cáo này sẽ phân tích sâu về dự án trò chơi RPG theo lượt đang được phát triển, với những tính năng nổi bật, quy trình phát triển và những thách thức mà phát triển đã vượt qua.

# CHƯƠNG 1: LÝ THUYẾT

## 1.1 Mục tiêu và phương pháp nghiên cứu

**-** Mục tiêu tổng quát: Xây dựng một game chiến thuật theo lượt (Turn-Based RPG) sử dụng Godot Engine 4 nhằm tạo ra sản phẩm hoàn chỉnh, thể hiện sự kết hợp giữa công nghệ và sáng tạo, đồng thời mang đến trải nghiệm người dùng hấp dẫn.

- Mục tiêu cụ thể:

+ Nghiên cứu các khái niệm cơ bản và yêu cầu đối với game chiến thuật theo lượt.

+ Lựa chọn công cụ phát triển phù hợp và triển khai hệ thống các thành phần chính trong game (ví dụ: hệ thống chiến đấu, giao diện người dùng, cốt truyện, và NPC).

+ Thiết kế các yếu tố đồ họa và âm thanh phù hợp, nâng cao tính thẩm mỹ và trải nghiệm của trò chơi.

+ Đánh giá hiệu quả của sản phẩm và cải tiến dựa trên thử nghiệm người dùng.

- Phương pháp nghiên cứu:

+ Nghiên cứu tài liệu: Tìm hiểu lý thuyết về game chiến thuật, các tài liệu liên quan đến phát triển game bằng Godot Engine.

+ Thực nghiệm: Triển khai và thử nghiệm các tính năng của game qua từng giai đoạn phát triển.

+ Phân tích và tổng hợp: Đánh giá, so sánh để cải thiện chức năng và trải nghiệm người chơi.

- Đối tượng nghiên cứu: Công cụ Godot Engine và các hệ thống liên quan (Animation, Dialog, Combat Arena, Inventory...).

- Vấn đề nghiên cứu:  
+ Làm thế nào để phát triển một hệ thống chiến đấu chiến thuật theo lượt hoạt động hiệu quả và trực quan?

+ Làm thế nào để tối ưu hóa cốt truyện và tương tác nhân vật trong game nhằm thu hút người chơi?

+ Những yếu tố nào trong thiết kế giao diện người dùng và đồ họa có thể nâng cao trải nghiệm chơi?

+ Làm thế nào để áp dụng các công cụ và phương pháp trong Godot Engine để tạo ra một game hoàn chỉnh?

## 1.2 Khái niệm về Game

Game là một hình thức giải trí tương tác, trong đó người chơi phải tuân theo một số quy tắc và mục tiêu nhất định để đạt được kết quả mong muốn. Game có thể diễn ra dưới nhiều hình thức khác nhau, từ các trò chơi truyền thống ngoài trời đến các trò chơi kỹ thuật số trên máy tính, máy chơi game, hoặc thiết bị di động. Trò chơi mang tính giải trí nhưng cũng có thể rèn luyện kỹ năng, sự khéo léo, tư duy chiến thuật, hoặc phản xạ nhanh.

Một game thường có những yếu tố cơ bản sau:

+ Mục tiêu: Là một yếu tố quan trọng trong bất kỳ trò chơi nào, vì nó xác định kết quả mà người chơi muốn đạt được. Mục tiêu có thể đa dạng, phụ thuộc vào loại game:

Game nhập vai (RPG): Người chơi có thể hướng tới mục tiêu tiêu diệt kẻ thù, hoàn thành nhiệm vụ, hoặc phát triển nhân vật của mình lên cấp độ cao hơn.

Game chiến thuật (Strategy): Mục tiêu có thể là tiêu diệt đối thủ, quản lý tài nguyên, hoặc giành được quyền kiểm soát lãnh thổ.

Game giải đố: Người chơi cần giải quyết các vấn đề hoặc câu đố để tiến lên cấp độ tiếp theo.

Mục tiêu của game không chỉ giúp tạo động lực cho người chơi mà còn định hướng cho các hành động của họ trong suốt quá trình chơi.

+ Luật chơi: Là những quy định và giới hạn mà người chơi phải tuân thủ trong suốt quá trình chơi. Các quy tắc này giúp duy trì sự công bằng và tạo ra cấu trúc cho game, giúp người chơi hiểu cách mà họ có thể tương tác với trò chơi và với các đối thủ khác. Trong các game chiến thuật, luật chơi có thể quy định người chơi chỉ có thể di chuyển một số lượng bước nhất định trong một lượt, hoặc chỉ có thể sử dụng một số loại quân lính nhất định trong một lượt. Có thể biến đổi hoặc nâng cấp theo cấp độ, tạo nên sự phong phú và thách thức mới cho người chơi.

+ Người chơi: Là trung tâm của mọi trò chơi không thể thiếu trong mọi game. Họ có thể là cá nhân hoặc nhóm người tham gia vào trò chơi, và họ phải tương tác với trò chơi để đạt được mục tiêu. Game có thể có một người chơi hoặc nhiều người chơi (đối kháng hoặc hợp tác).

Một người chơi (Single-player): Người chơi tương tác với game và hệ thống AI hoặc theo một cốt truyện cố định.

Nhiều người chơi (Multiplayer): Nhiều người chơi có thể cùng tham gia trong một trò chơi, ở chế độ đối kháng hoặc hợp tác.

Game co-op: Người chơi cùng hợp tác để đạt mục tiêu chung, thay vì đối kháng lẫn nhau.

+ Môi trường tương tác: Là không gian ảo nơi trò chơi diễn ra. Trong các trò chơi điện tử, môi trường này bao gồm giao diện đồ họa, nhân vật, và các yếu tố tương tác khác như vật phẩm, kẻ thù, hay các điều kiện môi trường. Môi trường tương tác tạo nên thế giới mà người chơi phải khám phá, đối đầu và tương tác trong suốt quá trình chơi.

Đồ họa: Giao diện, phong cảnh, các đối tượng và nhân vật được thể hiện trên màn hình. Đồ họa giúp tái tạo thế giới ảo và tạo ra sự sống động cho trải nghiệm chơi.

Các yếu tố tương tác: Bao gồm nhân vật, vật phẩm, kẻ thù, chướng ngại vật, hoặc các đối tượng khác mà người chơi có thể tương tác. Các yếu tố này thường giúp tạo ra thử thách hoặc hỗ trợ cho người chơi trong việc đạt mục tiêu.

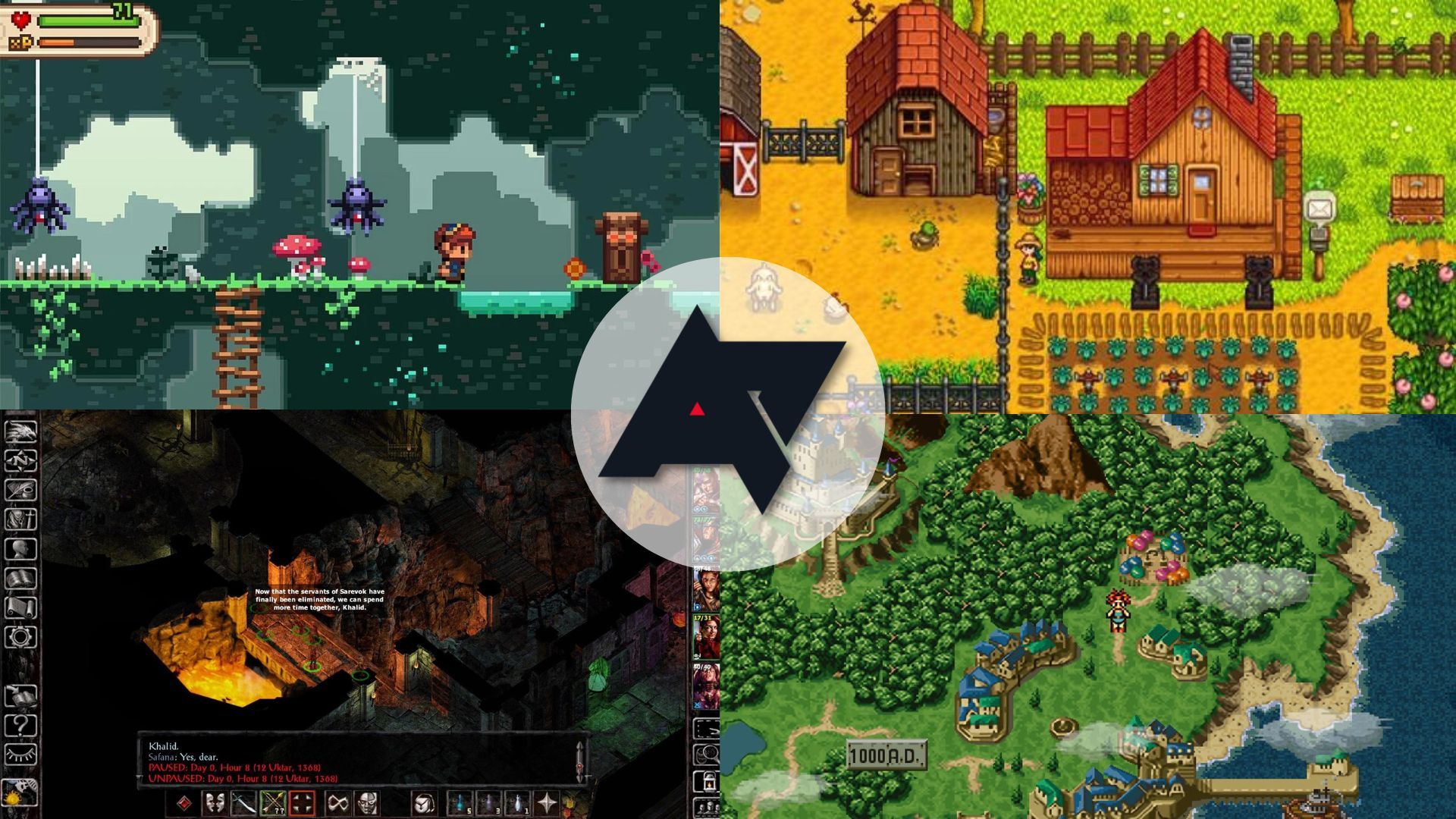
Môi trường âm thanh: Hiệu ứng âm thanh và nhạc nền cũng là một phần không thể thiếu, giúp tạo không khí và tăng cường sự hấp dẫn cho game.

Một game nhập vai có thể diễn ra trong một thế giới mở, nơi người chơi có thể tự do di chuyển, khám phá và tương tác với nhiều vật phẩm và nhân vật khác nhau.

+ Ngoài các yếu tố cơ bản trên, game còn có nhiều thể loại khác nhau, mỗi thể loại lại tập trung vào một kiểu chơi và trải nghiệm khác biệt. Một số thể loại game phổ biến bao gồm:

Game nhập vai (RPG - Role-playing game): Người chơi sẽ hóa thân vào nhân vật, phát triển các kỹ năng và thực hiện các nhiệm vụ dựa trên cốt truyện của game.





Hình 1.1: Hình ảnh game chiến thuật Honkai Star Rail và một số game indie phổ biến

Game hành động (Action games): Tập trung vào tốc độ phản xạ và kỹ năng điều khiển nhân vật, thường yêu cầu người chơi phải thực hiện các thao tác nhanh và chính xác.

Game chiến thuật (Strategy games): Tập trung vào khả năng tư duy chiến lược và quản lý tài nguyên, như game xây dựng thành phố, chiến tranh, hoặc quản lý quân đội.

Game thể thao: Mô phỏng các môn thể thao thực tế như bóng đá, bóng rổ, hoặc đua xe.

Game giải đố (Puzzle games): Người chơi phải giải các câu đố logic hoặc thực hiện các thử thách trí tuệ.



Hình 1.2:Trận chiến giữa quân đội Protoss và Terran trong game Starcraft I

+ Game không chỉ đơn thuần là giải trí mà còn là một môi trường tương tác phức tạp với nhiều yếu tố cơ bản như mục tiêu, luật chơi, người chơi, môi trường tương tác, và thách thức. Một tựa game hay không chỉ cần gameplay ấn tượng mà còn cần ý nghĩa, thông điệp mà game mang lại. Điển hình là đã có nhiều tựa game đơn giản chỉ bằng việc người chơi chủ yếu tương tác thông qua việc click chuột để tiến hành cốt truyện, đọc các đoạn đối thoại và lựa chọn quyết định trong các tình huống như Visual Novel (tiểu thuyết hình ảnh). Từ đó dẫn đến nhưng kết quả khác nhau nhưng lại được rất nhiều người ưa chuộng ví dụ như Steins;Gate, Doki Doki Literature Club, Clannad. Việc nắm rõ các yếu tố này giúp chúng ta có cái nhìn sâu sắc hơn về cấu trúc và cơ chế vận hành của game, từ đó ứng dụng trong việc phát triển hoặc trải nghiệm trò chơi một cách toàn diện và hiệu quả.



Hình 1.3: Một số game thể loại tương tác kể chuyện

## 1.3 Khái niệm và yêu cầu game chiến thuật

- Game chiến thuật (Strategy Game) là một thể loại trò chơi điện tử hoặc trò chơi vật lý đòi hỏi người chơi phải sử dụng tư duy chiến lược để lên kế hoạch, đưa ra quyết định và triển khai các hành động nhằm đạt được mục tiêu cuối cùng. Thể loại này thường nhấn mạnh vào yếu tố tư duy logic, phân tích tình huống, và quản lý tài nguyên thay vì chỉ dựa vào phản xạ nhanh hoặc kỹ năng thao tác.

Trong game chiến thuật, người chơi thường phải đối mặt với các tình huống phức tạp, cần phải cân nhắc kỹ lưỡng trước khi đưa ra quyết định, vì những hành động của họ sẽ ảnh hưởng trực tiếp đến kết quả của trò chơi. Các tựa game chiến thuật nổi tiếng bao gồm "Civilization", "StarCraft", "Total War", "XCOM", và "Fire Emblem".

Có hai dạng chính của game chiến thuật:

Game chiến thuật theo lượt (Turn-Based Strategy - TBS): Người chơi và đối thủ thực hiện các hành động theo lượt. Mỗi bên có thời gian riêng để suy nghĩ và đưa ra quyết định. Ví dụ như trò chơi "Final Fantasy Tactics" hay "Advance Wars".

Game chiến thuật thời gian thực (Real-Time Strategy - RTS): Các hành động diễn ra đồng thời và liên tục, yêu cầu người chơi vừa quản lý, vừa đưa ra quyết định một cách nhanh chóng. Ví dụ: "StarCraft", "Age of Empires".

- Một game chiến thuật thành công cần đáp ứng một loạt yêu cầu về thiết kế và phát triển nhằm đảm bảo rằng trò chơi sẽ mang lại trải nghiệm thú vị và thách thức cho người chơi. Dưới đây là những yêu cầu cơ bản của một game chiến thuật:

Cân bằng trong gameplay:

+ Đảm bảo sự cân bằng giữa các phe phái hoặc đơn vị trong trò chơi để không bên nào có lợi thế vượt trội một cách bất hợp lý. Sự cân bằng này giúp người chơi có cơ hội chiến thắng bằng cách dựa vào kỹ năng và chiến lược của mình.

Độ khó tăng dần:

+ Game chiến thuật cần có mức độ khó tăng dần để duy trì sự hấp dẫn và thách thức. Người chơi cần phải cảm thấy rằng trò chơi trở nên khó hơn khi họ tiến xa hơn, đòi hỏi họ phát triển kỹ năng và chiến thuật của mình.

Hệ thống ra quyết định:

+ Người chơi phải đối mặt với nhiều quyết định quan trọng trong suốt trò chơi. Những quyết định này có thể liên quan đến việc quản lý tài nguyên, chỉ huy quân đội, lựa chọn chiến thuật chiến đấu hoặc phát triển cơ sở hạ tầng. Các lựa chọn này sẽ ảnh hưởng trực tiếp đến kết quả của trò chơi.

Quản lý tài nguyên:

+ Trong nhiều game chiến thuật, quản lý tài nguyên là một yếu tố quan trọng. Người chơi phải thu thập, phân bổ và sử dụng các nguồn lực một cách hợp lý để xây dựng quân đội, công nghệ hoặc công trình mới, từ đó củng cố chiến lược của mình.

Chiến thuật và chiến lược:

+ Game cần cung cấp cho người chơi nhiều lựa chọn chiến thuật khác nhau để họ có thể áp dụng các chiến lược khác nhau tùy theo tình huống. Ví dụ, người chơi có thể chọn tấn công mạnh mẽ hoặc sử dụng chiến thuật phòng thủ tùy vào tình thế.

+ Hệ thống AI: Đối thủ do máy tính điều khiển (AI) phải có khả năng thách thức người chơi một cách hợp lý. AI phải biết ứng phó với các chiến thuật khác nhau của người chơi và tạo ra sự đa dạng trong gameplay.

+ Tính tương tác và hồi đáp: Người chơi cần phải tương tác thường xuyên với môi trường, đối thủ và các đơn vị của mình trong game. Các hành động của họ phải được phản hồi một cách nhanh chóng và rõ ràng, giúp họ điều chỉnh chiến thuật một cách hiệu quả.

+ Cốt truyện và bối cảnh hấp dẫn: Một game chiến thuật tốt không chỉ dựa trên gameplay mà còn cần một cốt truyện và bối cảnh hấp dẫn. Điều này giúp tạo ra động lực và mục tiêu cho người chơi, đồng thời nâng cao trải nghiệm chơi game.

+ Hệ thống nâng cấp: Ngoài việc quản lý tài nguyên, game chiến thuật thường yêu cầu người chơi đầu tư vào nghiên cứu hoặc phát triển công nghệ để cải thiện các đơn vị quân đội, vũ khí, và công trình.

## 1.4 Quy trình triển khai Game chiến thuật?

- Quy trình phát triển một game chiến thuật có thể được chia thành nhiều bước, từ giai đoạn lên ý tưởng ban đầu đến khi hoàn thiện và phát hành trò chơi. Dưới đây là các bước triển khai cụ thể:

+ Giai đoạn lên ý tưởng và thiết kế (Concept and Design Phase):

Ý tưởng ban đầu: Xác định ý tưởng chính cho game, bao gồm loại chiến thuật (theo lượt hay thời gian thực), bối cảnh, cốt truyện, và các yếu tố gameplay chính.

Thiết kế tổng thể: Xây dựng các yếu tố chính của trò chơi, bao gồm hệ thống chiến đấu, quản lý tài nguyên, nhiệm vụ, và các cơ chế chiến thuật. Đảm bảo rằng các yếu tố này tương tác với nhau một cách hợp lý.

Thiết kế chi tiết: Đưa ra các đặc điểm cụ thể của từng đơn vị, nhân vật, quân đội, tài nguyên, và địa hình. Thiết kế giao diện người dùng (UI) sao cho dễ sử dụng và phù hợp với phong cách game.

+ Giai đoạn phát triển (Development Phase):

Lập trình cơ chế chính: Bắt đầu với việc lập trình các cơ chế chính của trò chơi như hệ thống chiến đấu, điều khiển đơn vị, AI, và quản lý tài nguyên.

Xây dựng AI: Phát triển AI để tạo ra các đối thủ thông minh và có thể thách thức người chơi. AI cần có khả năng ứng phó với các chiến thuật khác nhau và điều chỉnh hành vi dựa trên tình huống.

Phát triển đồ họa và âm thanh: Tạo ra các yếu tố đồ họa như nhân vật, bản đồ, và hiệu ứng đặc biệt. Đồng thời phát triển hệ thống âm thanh để tăng cường trải nghiệm chơi game, bao gồm âm nhạc nền và hiệu ứng âm thanh.

+ Giai đoạn thử nghiệm (Testing Phase):

Kiểm tra lỗi (Bug Testing): Thử nghiệm trò chơi để tìm ra các lỗi (bugs) và vấn đề trong gameplay. Đảm bảo rằng các cơ chế hoạt động như mong đợi và không có lỗi nghiêm trọng làm gián đoạn trải nghiệm của người chơi.

Cân bằng game (Balance Testing): Điều chỉnh các yếu tố trong trò chơi để đảm bảo sự cân bằng giữa các đơn vị, tài nguyên và chiến lược. Điều này rất quan trọng để tránh việc trò chơi trở nên quá dễ hoặc quá khó.

Giai đoạn hoàn thiện và phát hành (Polishing and Release Phase):

Tinh chỉnh cuối cùng: Tinh chỉnh các yếu tố đồ họa, âm thanh, và giao diện để đảm bảo trò chơi đạt được mức độ hoàn thiện cao nhất. Đảm bảo rằng tất cả các yếu tố đều hòa hợp với nhau và tạo ra một trải nghiệm chơi game liền mạch.

Chuẩn bị phát hành: Hoàn thiện phiên bản cuối cùng của game, đóng gói và chuẩn bị phát hành trên các nền tảng khác nhau. Tiến hành các chiến dịch marketing và quảng bá để giới thiệu game đến với cộng đồng người chơi.

+ Giai đoạn hậu phát hành (Post-Release Phase):

Cập nhật và vá lỗi: Sau khi game được phát hành, tiếp tục theo dõi phản hồi từ người chơi để sửa lỗi và cải tiến gameplay. Phát hành các bản cập nhật để giữ cho trò chơi luôn mới mẻ và hấp dẫn.

Mở rộng nội dung: Có thể triển khai thêm các bản mở rộng (DLCs) hoặc cập nhật nội dung mới để kéo dài tuổi thọ của game và duy trì sự quan tâm từ người chơi.

## 1.5 Lựa chọn công cụ phát triển

- Unity: Unity là một trong những công cụ phát triển game phổ biến nhất hiện nay và được sử dụng rộng rãi trong ngành công nghiệp game. Điểm mạnh của Unity là hỗ trợ tốt cho cả game 2D và 3D, cùng với thư viện asset phong phú, cho phép các nhà phát triển dễ dàng tích hợp và xây dựng các dự án đa dạng. Unity sử dụng ngôn ngữ lập trình C#, đây là một ngôn ngữ mạnh mẽ và dễ học, đặc biệt đối với những người đã có kinh nghiệm lập trình trước đó.

Unity còn nổi bật bởi khả năng xuất bản trò chơi đa nền tảng, từ máy tính (Windows, macOS, Linux) cho đến thiết bị di động (Android, iOS) và thậm chí cả console (PS4, Xbox, Nintendo). Điều này giúp các nhà phát triển dễ dàng tiếp cận một lượng lớn người chơi mà không phải viết lại mã nguồn cho từng nền tảng riêng biệt.

Tuy nhiên, Unity cũng có những hạn chế. Thứ nhất, nó đòi hỏi người phát triển phải có kiến thức rộng về C# và phải làm quen với các hệ sinh thái của Unity, điều này có thể gây khó khăn cho người mới bắt đầu. Hơn nữa, việc sử dụng Unity yêu cầu một khoản chi phí nhất định khi dự án đạt đến quy mô lớn hoặc khi cần xuất bản trên các nền tảng thương mại. Điều này có thể là một rào cản với những nhà phát triển game độc lập có ngân sách hạn chế.

- Unreal Engine: Unreal Engine, được phát triển bởi Epic Games, là một trong những công cụ mạnh mẽ nhất để phát triển các trò chơi 3D với chất lượng đồ họa tuyệt vời. Unreal Engine sử dụng ngôn ngữ C++ và được tích hợp các công cụ tối ưu hóa đồ họa và âm thanh, cho phép tạo ra các trò chơi có chất lượng cao. Đây là một lựa chọn lý tưởng cho những dự án game 3D đòi hỏi hình ảnh sống động, hiệu ứng phức tạp và trải nghiệm chân thực.

Unreal Engine có một cộng đồng rộng lớn và cung cấp các khóa học, tài liệu học tập chính thức, giúp người dùng nhanh chóng tiếp cận và nắm vững các kỹ năng cần thiết. Unreal cũng hỗ trợ rất nhiều tính năng tiên tiến như hệ thống vật lý, ánh sáng, đổ bóng, và hoạt ảnh xương (skeletal animation).

Tuy nhiên, hạn chế lớn nhất của Unreal Engine là yêu cầu phần cứng rất cao, vì vậy việc phát triển và thử nghiệm trên các máy tính có cấu hình thấp sẽ gặp khó khăn. Đồng thời, học ngôn ngữ C++ và làm quen với hệ sinh thái của Unreal có thể mất nhiều thời gian, đặc biệt đối với những người chưa có kinh nghiệm lập trình hoặc đang bắt đầu với lĩnh vực phát triển game.

- Construct: Construct là một công cụ phát triển game 2D không yêu cầu kỹ năng lập trình. Đây là lựa chọn phù hợp cho những người mới bắt đầu, hoặc những nhà phát triển không chuyên muốn tạo ra các trò chơi 2D đơn giản mà không cần phải viết mã. Construct cung cấp một giao diện kéo-thả trực quan và hỗ trợ tạo ra các trò chơi một cách nhanh chóng.

Construct có điểm mạnh là dễ học và thân thiện với người mới bắt đầu, giúp người dùng nhanh chóng tạo ra các trò chơi 2D đơn giản với cơ chế logic không quá phức tạp. Tuy nhiên, Construct không phải là một lựa chọn tốt để phát triển các trò chơi phức tạp hoặc 3D, vì nó bị giới hạn trong việc xây dựng các tính năng và khả năng tùy biến.

Từ đó em lựa chọn Godot Engine là một công cụ phù hợp cho việc phát triển đề tài. Bởi engine này hỗ trợ ngôn ngữ gdscript là một ngôn ngữ script tương tự dựa trên nền tảng python, C++, C# những ngôn ngữ được học trong suốt quá trình học tập tại trường. Ngoài ra còn được sự trợ giúp mạnh mẽ bởi cộng đồng sôi động mang lại. So với Unreal Engine hoặc Unity, Godot có dung lượng nhỏ gọn hơn và yêu cầu phần cứng thấp hơn, rất phù hợp với những máy tính có cấu hình vừa phải. Giao diện của Godot cũng khá trực quan, giúp người mới dễ dàng học và bắt đầu.

Godot Engine là một nền tảng phát triển game mã nguồn mở và miễn phí, được sử dụng rộng rãi để phát triển các trò chơi 2D và 3D. Được phát triển lần đầu tiên bởi Juan Linietsky và Ariel Manzur vào năm 2007, Godot đã nhanh chóng trở thành một trong những công cụ phát triển game phổ biến nhất nhờ vào sự linh hoạt, tính dễ sử dụng, và sự hỗ trợ mạnh mẽ từ cộng đồng mã nguồn mở. Dưới đây là một cái nhìn tổng quan về những đặc điểm nổi bật của Godot Engine:

- Đa Nền Tảng (Cross-Platform)

Godot hỗ trợ xuất bản trò chơi lên nhiều nền tảng khác nhau bao gồm Windows, macOS, Linux, Android, iOS, HTML5, và nhiều hệ máy console khác. Điều này cho phép các nhà phát triển tiếp cận với một lượng lớn người chơi mà không cần phải viết lại mã nguồn cho từng nền tảng riêng biệt.

- Hỗ Trợ Cả 2D và 3D

Godot cung cấp các công cụ mạnh mẽ cho cả phát triển game 2D và 3D. Đặc biệt, hệ thống 2D của Godot là một trong những điểm mạnh lớn nhất, với các tính năng như:

Hệ tọa độ màn hình thực tế: giúp dễ dàng xử lý các yếu tố đồ họa và vật lý trong môi trường 2D.

Hệ thống sprite và animation: rất linh hoạt và cho phép tạo ra các chuyển động mượt mà với các khung hình định trước.

Hỗ trợ light và shadow: trong môi trường 2D, điều này thường không phổ biến trong các công cụ khác.

Về 3D, Godot cung cấp một hệ thống kết xuất đồ họa mạnh mẽ với khả năng hỗ trợ PBR (Physically Based Rendering), hệ thống ánh sáng và bóng đổ tiên tiến, cùng với hỗ trợ cho vật lý 3D và hoạt ảnh xương (skeletal animation).

- Ngôn Ngữ Lập Trình Linh Hoạt

Godot hỗ trợ nhiều ngôn ngữ lập trình, trong đó ngôn ngữ chính là GDScript, một ngôn ngữ kịch bản tương tự như Python, dễ học và rất thích hợp cho người mới bắt đầu. Ngoài ra, Godot còn hỗ trợ VisualScript (một ngôn ngữ lập trình trực quan dựa trên các nút), C# (nhờ tích hợp Mono), và C++ (đối với các tác vụ yêu cầu hiệu suất cao).

- Hệ Thống Node-Based

Một trong những đặc điểm nổi bật của Godot là hệ thống dựa trên các node (nút). Mọi thứ trong Godot đều được cấu trúc dưới dạng các node, và các node này có thể được kết hợp lại để tạo ra các đối tượng phức tạp hơn. Ví dụ, một nhân vật trong game có thể là một node chính, chứa các node con như sprite, collider, và script để quản lý hành vi.

Hệ thống node này giúp việc tổ chức và quản lý dự án trở nên dễ dàng hơn, đồng thời cho phép sự tái sử dụng mã nguồn và các tài nguyên khác một cách hiệu quả.

- Giao Diện Người Dùng Trực Quan

Godot có giao diện người dùng thân thiện và trực quan, cho phép các nhà phát triển dễ dàng kéo thả các tài nguyên, tổ chức các scene (cảnh) và đối tượng, cũng như tạo các kết nối giữa các node. Các công cụ như trình biên tập scene, trình biên tập script, và các cửa sổ kiểm tra tài sản đều được tích hợp trong một giao diện thống nhất, giúp tăng tốc độ phát triển.

- Mã Nguồn Mở và Cộng Đồng Mạnh Mẽ

Là một phần mềm mã nguồn mở được cấp phép dưới MIT License, Godot cho phép bất kỳ ai cũng có thể truy cập và sửa đổi mã nguồn của nó. Điều này đã tạo ra một cộng đồng phát triển mạnh mẽ, luôn đóng góp các cải tiến và tính năng mới. Ngoài ra, cộng đồng Godot rất năng động, với nhiều tài liệu, hướng dẫn, và hỗ trợ từ các diễn đàn và nhóm phát triển.

- Nhẹ và Hiệu Quả

Godot được thiết kế để có dung lượng nhỏ gọn và sử dụng tài nguyên hệ thống hiệu quả. Điều này giúp cho Godot có thể chạy mượt mà trên các hệ thống phần cứng hạn chế, đồng thời cho phép các nhà phát triển triển khai game của mình trên các nền tảng với yêu cầu tài nguyên thấp.

- Hỗ Trợ Asset Pipeline Tích Hợp

Godot tích hợp sẵn các công cụ hỗ trợ quản lý và tối ưu hóa tài nguyên (assets) như hình ảnh, âm thanh, và mô hình 3D. Các công cụ này cho phép các nhà phát triển nhập, xuất, và tối ưu hóa tài nguyên trực tiếp trong Godot, mà không cần phải sử dụng phần mềm bên ngoài.

## 1.6 Đối tượng tham gia hệ thống

Godot Engine là một nền tảng phát triển game với mã nguồn mở, và nó được thiết kế để phục vụ một loạt các đối tượng khác nhau trong ngành công nghiệp phát triển game. Các đối tượng tham gia hệ thống bao gồm:

1. Nhà Phát Triển Game Độc Lập (Indie Developers)

Godot được biết đến rộng rãi trong cộng đồng các nhà phát triển game độc lập (indie) vì tính linh hoạt, chi phí bằng 0 (miễn phí), và khả năng hỗ trợ đa nền tảng. Những nhà phát triển này thường làm việc trên các dự án nhỏ hoặc vừa, và Godot cung cấp cho họ một công cụ mạnh mẽ để thực hiện ý tưởng game của mình mà không cần phải đầu tư nhiều vào công nghệ hay phần mềm đắt tiền.

2. Công Ty Phát Triển Game Nhỏ và Vừa

Các công ty phát triển game có quy mô nhỏ và vừa thường lựa chọn Godot vì nó cho phép họ xây dựng các dự án game chất lượng cao với chi phí phát triển thấp. Ngoài ra, việc sử dụng Godot giúp các công ty này tránh được chi phí cấp phép mà họ có thể gặp phải với các engine thương mại khác.

3. Giáo Viên và sinh viên

Godot là một công cụ tuyệt vời cho việc giảng dạy và học tập trong các trường học và viện đào tạo. Giáo viên có thể sử dụng Godot để dạy các khái niệm cơ bản về phát triển game và lập trình, trong khi có thể dễ dàng truy cập và sử dụng công cụ này để phát triển các dự án học tập hoặc cá nhân.

4. Người Dùng Không Chuyên (Hobbyists)

Những người yêu thích phát triển game như một sở thích cá nhân cũng là một phần quan trọng trong cộng đồng người dùng Godot. Họ có thể là những người mới bắt đầu hoặc những nhà phát triển có kinh nghiệm muốn thử nghiệm các ý tưởng mới mà không phải đầu tư quá nhiều thời gian hoặc tiền bạc.

5. Cộng Đồng Mã Nguồn Mở

Cộng đồng mã nguồn mở đóng vai trò rất quan trọng trong sự phát triển và duy trì Godot. Những lập trình viên, nhà thiết kế, và những người đóng góp khác từ khắp nơi trên thế giới tham gia vào việc cải thiện và mở rộng tính năng của Godot. Họ cũng tạo ra các tài liệu, hướng dẫn, và hỗ trợ các thành viên mới trong cộng đồng.

## 1.7 Lịch sử hình thành của Godot Engine

Godot Engine có một lịch sử hình thành độc đáo, bắt đầu từ một dự án nội bộ và dần dần trở thành một trong những engine phát triển game mã nguồn mở phổ biến nhất thế giới.

1. Khởi Đầu (2007-2014)

Godot Engine được tạo ra bởi Juan Linietsky và Ariel Manzur vào năm 2007 như một công cụ phát triển game nội bộ cho một công ty phát triển game tại Argentina. Ban đầu, Godot được sử dụng để tạo ra các trò chơi thương mại và không được công bố ra công chúng.

Trong những năm đầu, Juan Linietsky và Ariel Manzur đã tiếp tục phát triển và cải tiến engine này, tập trung vào việc tạo ra một công cụ phát triển game có khả năng hỗ trợ cả 2D và 3D một cách hiệu quả.

2. Công Bố Mã Nguồn Mở (2014)

Đến năm 2014, Juan Linietsky và Ariel Manzur quyết định phát hành Godot dưới dạng mã nguồn mở với giấy phép MIT. Quyết định này nhằm mục đích tạo ra một công cụ phát triển game chất lượng cao có thể tiếp cận được với tất cả mọi người, không phân biệt quy mô dự án hay nguồn tài chính.

Việc phát hành mã nguồn mở đã thu hút được sự chú ý từ cộng đồng lập trình viên và nhà phát triển game trên toàn thế giới, và từ đó Godot bắt đầu phát triển với sự đóng góp của nhiều người.

3. Phát Triển và Tăng Trưởng (2014-Nay)

Sau khi trở thành mã nguồn mở, Godot đã nhận được sự hỗ trợ và đóng góp từ cộng đồng mã nguồn mở toàn cầu. Điều này giúp engine liên tục được cập nhật và cải tiến với nhiều tính năng mới, bao gồm cả hệ thống kết xuất đồ họa 3D nâng cao, hỗ trợ PBR, và nhiều công cụ hỗ trợ phát triển khác.

Godot cũng nhanh chóng trở nên phổ biến nhờ vào khả năng dễ sử dụng, tính linh hoạt, và sự hỗ trợ từ cộng đồng. Cộng đồng Godot đã tạo ra hàng loạt tài liệu, hướng dẫn, và công cụ phụ trợ để hỗ trợ các nhà phát triển mới và kỳ cựu.

4. Godot 3.x và Sự Cải Tiến Toàn Diện

Godot 3.0, phát hành vào tháng 1 năm 2018, là một bước ngoặt quan trọng trong lịch sử của Godot. Phiên bản này giới thiệu nhiều tính năng tiên tiến như hỗ trợ VR/AR, hệ thống kết xuất đồ họa dựa trên vật lý (PBR), và cải tiến đáng kể về hiệu suất.

Kể từ đó, các phiên bản Godot 3.x liên tiếp được phát hành với nhiều cải tiến, và cộng đồng Godot đã phát triển mạnh mẽ, không chỉ trong việc phát triển game mà còn trong các dự án giáo dục và nghiên cứu.

5. Godot 4.0 và Tương Lai

Godot 4.0, đang được phát triển và dự kiến ra mắt trong tương lai gần, hứa hẹn mang đến nhiều cải tiến lớn, đặc biệt trong lĩnh vực đồ họa 3D và hỗ trợ Vulkan. Điều này thể hiện tham vọng của Godot trong việc cạnh tranh với các engine thương mại hàng đầu như Unity và Unreal Engine.

Với sự phát triển không ngừng và sự hỗ trợ mạnh mẽ từ cộng đồng, Godot Engine tiếp tục khẳng định vị thế của mình như một trong những công cụ phát triển game mã nguồn mở hàng đầu thế giới.

## 1.8 Các thành phần trong Godot

Godot Engine là một công cụ phát triển game toàn diện với nhiều thành phần và tính năng hỗ trợ cho cả phát triển game 2D và 3D. Dưới đây là tổng quan về các thành phần chính của Godot:

- Scene và Node

Scene (Cảnh): Trong Godot, mọi thứ đều bắt đầu từ một scene. Scene là đơn vị cơ bản nhất trong Godot, chứa các đối tượng như nhân vật, đối tượng môi trường, giao diện người dùng, và các yếu tố khác. Một scene có thể là một phần của game (như một level hoặc một màn hình chính) hoặc một đối tượng phức tạp có thể được tái sử dụng trong các scene khác.

Node (Nút): Các scene trong Godot được tạo thành từ các node. Node là các đối tượng cơ bản và mỗi node thực hiện một chức năng cụ thể như hiển thị hình ảnh, xử lý âm thanh, quản lý đầu vào, hoặc thực thi logic trò chơi. Có hơn 100 loại node khác nhau trong Godot, cho phép nhà phát triển xây dựng cấu trúc game theo cách linh hoạt và mạnh mẽ. Có hệ thống scene-based mạnh mẽ, giúp dễ dàng tổ chức và tái sử dụng các phần của trò chơi. Điều này giúp tiết kiệm thời gian phát triển và quản lý dự án hiệu quả hơn. Miễn phí và không yêu cầu tài khoản như Unity hay Unreal Engine phù hợp với việc học hỏi và nghiên cứu.

- GDScript và Ngôn Ngữ Lập Trình

GDScript: Đây là ngôn ngữ lập trình chính được sử dụng trong Godot. GDScript được thiết kế để tích hợp chặt chẽ với Godot, dễ học và có cú pháp tương tự như Python. Nó cho phép nhà phát triển viết script để điều khiển hành vi của các node và quản lý logic của trò chơi.

C#: Godot cũng hỗ trợ ngôn ngữ C# thông qua tích hợp với Mono. Điều này rất hữu ích cho các nhà phát triển đã quen thuộc với C# và muốn tận dụng sức mạnh của ngôn ngữ này trong các dự án Godot.

VisualScript: Đây là một ngôn ngữ lập trình trực quan cho phép tạo các kịch bản game thông qua giao diện kéo-thả, thay vì viết mã lệnh. Nó đặc biệt hữu ích cho những người mới bắt đầu hoặc những người không có kinh nghiệm lập trình.

C++: Đối với các tác vụ yêu cầu hiệu suất cao, Godot cho phép sử dụng C++ để viết các module hoặc phần mở rộng.

- Hệ Thống SceneTree

SceneTree: Là cấu trúc cây quản lý toàn bộ các scene và node trong trò chơi. Nó theo dõi các node hiện tại đang hoạt động, xử lý các tín hiệu (signals), và quản lý các vòng lặp game (game loops). SceneTree là yếu tố cốt lõi giúp Godot thực thi các hoạt động của trò chơi theo thời gian thực.

- Tín Hiệu (Signals)

Signals: Là hệ thống gửi và nhận sự kiện (event) giữa các node. Thay vì kiểm tra trạng thái liên tục (polling), Godot sử dụng hệ thống tín hiệu để các node có thể phát ra và lắng nghe các sự kiện xảy ra, giúp code trở nên dễ quản lý và hiệu quả hơn.

- Hệ Thống Animation

AnimationPlayer: Godot cung cấp một hệ thống animation mạnh mẽ cho phép tạo và quản lý các chuyển động của đối tượng trong game. AnimationPlayer có thể được sử dụng để tạo các hoạt ảnh phức tạp như chuyển động nhân vật, hiệu ứng, và các thay đổi giao diện người dùng.

AnimationTree: Là một công cụ mạnh mẽ để kết hợp và điều khiển nhiều animation một cách mượt mà. Nó cho phép quản lý trạng thái hoạt động của nhân vật (như đi, chạy, nhảy) và chuyển đổi giữa các trạng thái này.

- Giao Diện Người Dùng (UI)

Control Nodes: Godot có các node điều khiển (Control nodes) đặc biệt để xây dựng giao diện người dùng (UI). Các node này bao gồm Button, Label, Panel, TextEdit, và nhiều thành phần UI khác, giúp dễ dàng tạo và quản lý các giao diện người dùng phức tạp.

Theme và Customization: Godot cho phép tùy biến giao diện người dùng thông qua hệ thống themes. Người dùng có thể tạo ra các giao diện nhất quán với phong cách game của họ mà không cần viết code.

- Hệ Thống Physics (Vật Lý)

Physics2D và Physics3D: Godot có hệ thống vật lý tích hợp cho cả game 2D và 3D. Các hệ thống này hỗ trợ va chạm, trọng lực, và nhiều tính năng vật lý khác cần thiết cho một game thực tế.

Collision Detection: Godot cung cấp nhiều loại node cho phát hiện va chạm, như Area2D, RigidBody2D, và CollisionShape2D (cho 2D) và các tương đương của chúng cho 3D. Các node này giúp xác định khi nào và ở đâu va chạm xảy ra trong game.

- Audio (Âm Thanh)

AudioStreamPlayer: Được sử dụng để phát âm thanh trong game. Godot hỗ trợ phát âm thanh 2D và 3D, giúp tạo ra các hiệu ứng âm thanh chân thực, như tiếng bước chân di chuyển gần xa, hoặc tiếng động vòm trong không gian 3D.

Audio Buses: Godot cung cấp hệ thống bus âm thanh, cho phép xử lý và điều chỉnh âm thanh theo các cách phức tạp, bao gồm thay đổi âm lượng, thêm hiệu ứng, hoặc điều khiển âm thanh một cách chi tiết.

- Tài Nguyên (Resources)

Resource: Là các đối tượng dữ liệu cơ bản trong Godot, như hình ảnh, âm thanh, hoặc mô hình 3D. Các tài nguyên có thể được tải lên và sử dụng bởi các node khác trong game.

Importing Assets: Godot có hệ thống nhập tài nguyên tích hợp, hỗ trợ nhiều định dạng khác nhau và cho phép nhà phát triển tối ưu hóa tài nguyên trước khi sử dụng trong game.

- Hệ Thống Navigation (Điều Hướng)

Navigation2D và Navigation3D: Các hệ thống điều hướng cho phép tạo ra các tuyến đường (paths) mà nhân vật có thể di chuyển một cách tự động, tránh các chướng ngại vật trong môi trường 2D hoặc 3D.

NavigationMesh: Được sử dụng trong 3D, cho phép tạo ra một lưới điều hướng dựa trên hình dạng của cảnh, giúp nhân vật di chuyển mượt mà qua các địa hình phức tạp.

- Exporting (Xuất Bản)

Xuất Bản Đa Nền Tảng: Godot cho phép xuất bản game sang nhiều nền tảng bao gồm PC (Windows, macOS, Linux), Mobile (Android, iOS), Web (HTML5), và console. Quá trình xuất bản được tối ưu hóa để đảm bảo game hoạt động tốt trên từng nền tảng.

# CHƯƠNG 2: ỨNG DỤNG XÂY DỰNG MÔ HÌNH TRÒ CHƠI

## 2.1 Mô tả bài toán

- Tựa Game mà project này hướng đến sẽ có 2 thành phần chính cấu thành nên trải nghiệm chính cho người chơi: Thế giới(Cốt truyện) và Giao Tranh(Gameplay).

Thế giới (Overworld) là trải nghiệm của người chơi khi không trong giao tranh. Tại đây, người chơi sẻ được trải nghiệm tính nhập vai cũng như hầu hết các yếu tố cốt truyện của game. Người chơi có thể khám phá, tương tác với các đồ vật/nhân vật, thực hiện các cuộc hội thoại, nhận nhiệm vũ, hoàn thành nhiệm vụ…

Các bài toán về lập trình:

* Tạo hệ thống di chuyển 4/8 hướng xung quanh thế giới, dịch chuyển giữa các màn trong thế giới
* Tạo hệ thống vật lý (tường chặn di chuyển, thay đổi tốc độ di chuyển,...)
* Tạo hệ thống hội thoại với các nhân vật, có khả năng rẽ nhánh, cho người dùng lựa chọn dòng hội thoại theo ý muốn và phản hồi tương ứng
* Tạo hệ thống quản lý các nhiệm vụ.
* Tạo hệ thống chỉ số chi tiết của nhân vật và hệ thống nâng cấp nhân vật
* Tạo chức năng cài đặt cho phép người chơi chỉnh sửa theo ý muốn
* Tạo chức năng tương tác với đồ vật, minigame phụ để đa dạng hóa trải nghiệm
* Tạo chức năng Cutscene phục vụ cho cốt truyện
* Tạo chức năng Kho Chứa Đồ (Inventory) hỗ trợ việc xây dựng nhân vật và hoàn thành nhiệm vụ
* Các yêu cầu về đồ họa:
* Tạo GUI cho các chức năng cơ bản (Menu, Kho đồ, Cài đặt,...)
* Tạo các đồ họa cho các màn cấu thành thế giới
* Tạo đồ họa cho các nhân vật khi di chuyển trong thế giới
* Tạo đồ họa cho các hiệu ứng đặc biệt khác
* Các yêu về âm thanh:
* Tạo nhạc nền phù hợp với từng màn cụ thể
* Tạo các hiệu ứng âm thanh cho các hành động / sự kiện xảy ra trên thế giới

Kiểu giao tranh mà project này hướng tới được lấy cảm hứng từ series Epic Battle Fantasy.

Các bài toán về lập trình:

* Tạo hệ thống di chuyển trong giao tranh cho cả người chơi và máy tính, cùng với các kỹ năng có yếu tố di chuyển
* Tạo hệ thống kỹ năng được người chơi và máy sử dụng
* Tạo hệ thống quản lý Trạng thái game (State Machine)
* Tạo hệ thống AI điều khiển các kẻ địch, có thể có nhiều mức độ khó khăn
* Tạo hệ thống các chỉ số được sử dụng trong giao tranh(Máu, tấn công, phòng thủ, …)
* Tạo hệ thống các hiệu ứng đặc biệt(Hiệu ứng tăng/giảm chỉ số, hiệu ứng khống chế,...)
* Tạo hệ thống phần thưởng sau giao tranh dựa trên kẻ địch bị hạ gục
* Các yêu cầu về đồ họa:
* Tạo Background cho trận đấu
* Tạo đồ họa và animation cho các nhân vật trong giao tranh
* Tạo đồ họa và animation cho các kỹ năng trong giao tranh
* Tạo GUI để truyền tải các thông tin quan trọng (Máu, Sát thương, kỹ năng, hiệu ứng,...)
* Tạo đồ họa cho các hiệu ứng
* Các yêu cầu về âm thanh:
* Tạo nhạc nền cho giao tranh
* Tạo các hiệu ứng âm thanh trong giao tranh (Hiệu ứng kỹ năng, nhân vật,...)

- Người chơi sẽ nhập vai vào một tổ đội gồm pháp sư và chiến binh để khám phá thế giới, đánh quái và đối đầu với các boss. Người chơi sẽ chọn kỹ năng thi triển sao cho phù hợp để chiến thắng, vượt qua các vật cản. Đồng thời, người chơi có thể mua vật phẩm và thực hiện nhiệm vụ để tiến tới mục tiêu cuối cùng.

## 2.2 Nội dung của trò chơi

Bối cảnh: Trong một thế giới huyền ảo bị chia cắt bởi các phe phái tranh giành quyền lực, bạn nhập vai một anh hùng bị kéo vào cuộc chiến giữa các lực lượng hắc ám và ánh sáng. Với nhiệm vụ khôi phục lại hòa bình cho vương quốc, người chơi sẽ du hành qua các vùng đất, chiến đấu với kẻ thù, kết bạn với những đồng minh và khám phá những bí ẩn cổ xưa.

Cốt truyện: Vương quốc Aldoria từng là một vùng đất hòa bình, nhưng một thế lực tà ác đã thức tỉnh và gieo rắc sự hủy diệt khắp nơi. Bạn vào vai một chiến binh trẻ tuổi, được tiên tri là người mang trong mình sức mạnh để đối đầu với bóng tối. Hành trình bắt đầu tại ngôi làng nhỏ Aldorf, nơi bạn gặp những người bạn đồng hành đầu tiên và từ đó bước vào cuộc hành trình gian nan chống lại đội quân của Chúa tể Bóng Tối.  
Nhân vật chính:

Người chơi: Chiến binh với khả năng chiến đấu và Pháp sư có các kỹ năng phép thuật.

A group of people in a video game

Description automatically generatedA screenshot of a video game

Description automatically generated

Hình 2.1:Sprite sheet của nhân vật với các tư thế và hành động khác nhau.

NPC (nhân vật không điều khiển): Gồm những người dân, thầy phù thủy có thể giao tiếp, nhận nhiệm vụ.

+ Eldor the Wise - Thầy phù thủy già:

Eldor là một phù thủy già khôn ngoan sống tại làng Aldorf, nơi khởi đầu cuộc hành trình. Ông là người hướng dẫn người chơi trong những bước đầu tiên và cung cấp những thông tin quan trọng về lịch sử của thế lực bóng tối. Eldor cũng giúp mở khóa kỹ năng phép thuật cho nhân vật pháp sư, đồng thời trao cho người chơi những nhiệm vụ quan trọng liên quan đến việc thu thập các vật phẩm cổ xưa.

+ Lyra the Rogue - Nữ trộm khéo léo:

Lyra là một nhân vật bí ẩn mà người chơi sẽ gặp trong khu rừng xanh. Cô có kỹ năng trộm cắp và am hiểu về các cạm bẫy trong thế giới trò chơi. Ban đầu cô là một đối thủ, nhưng sau khi người chơi giúp cô thoát khỏi tay bọn quái vật trong một nhiệm vụ, cô trở thành đồng minh và cung cấp cho người chơi các nhiệm vụ liên quan đến việc đột nhập các tòa lâu đài của kẻ thù.

Vai trò: Đồng minh cung cấp thông tin và nhiệm vụ.

+ Lord Arthus - Lãnh chúa vùng tuyết:

Arthus là lãnh chúa của vùng đất lạnh giá phía bắc. Ông ta giữ một mảnh vũ khí quyền năng cần thiết để tiêu diệt Chúa tể Bóng Tối. Tuy nhiên, để nhận được sự giúp đỡ của ông, người chơi cần chứng minh lòng dũng cảm của mình bằng cách hoàn thành một số nhiệm vụ khắc nghiệt trong điều kiện lạnh giá của vùng tuyết.

Vai trò: Người giữ chìa khóa quan trọng trong cốt truyện, trao nhiệm vụ khó khăn cho người chơi.

- Lối chơi:

Người chơi sẽ khám phá các vùng đất khác nhau thông qua hệ thống di chuyển tự do hoặc theo lưới bản đồ. Trong quá trình đó, bạn sẽ thu thập tài nguyên, hoàn thành các nhiệm vụ phụ và chính, đồng thời nâng cấp và trang bị cho nhân vật.

Các vùng đất chính:

Rừng xanh: Khu vực rậm rạp, đầy rẫy quái vật và thử thách liên quan đến địa hình hiểm trở.

Vùng tuyết: Đối mặt với khí hậu khắc nghiệt và các sinh vật sống trong môi trường lạnh giá.

Sa mạc: Vùng đất khô cằn, đầy nguy hiểm từ những sinh vật sa mạc và bẫy cát.

Hệ thống chiến đấu: Làm theo phong cách chiến thuật theo lượt. Bạn cần chọn vị trí cho nhân vật của mình trên bản đồ và sử dụng các kỹ năng tấn công, phép thuật, hoặc hỗ trợ để đánh bại kẻ thù.

Tương tác với NPC: Bạn có thể thực hiện đối thoại, nhận nhiệm vụ từ NPC hoặc thực hiện các tương tác khác như trao đổi vật phẩm, tham gia các sự kiện trong game.

- Các tính năng:

Hệ thống đối thoại: Tương tác với NPC thông qua đối thoại, có thể thay đổi diễn biến câu chuyện tùy vào các lựa chọn trong game.

Hệ thống trang bị: Người chơi có thể thu thập và nâng cấp vũ khí, áo giáp, và các vật phẩm hỗ trợ khác.

Hệ thống kỹ năng: Người chơi có thể mở khóa và nâng cấp các kỹ năng chiến đấu và phép thuật qua các cấp độ.

## 2.3 Đầu vào của hệ thống game

### 2.3.1 Đầu vào từ người chơi

Tương tác của người chơi: Người chơi có thể chọn các kỹ năng, mục tiêu để tấn công, sử dụng vật phẩm trong kho đồ, và di chuyển nhân vật. Các thao tác này sẽ ảnh hưởng đến trạng thái của nhân vật và trận đấu.

Cài đặt cá nhân hóa: Người chơi có thể điều chỉnh các thiết lập như âm thanh, đồ họa, phù hợp với sở thích cá nhân để tăng trải nghiệm game.

### 2.3.2 Đầu vào từ AI và hệ thống

AI đối thủ: Hệ thống sẽ xử lý các hành động của AI, như việc lựa chọn kỹ năng và mục tiêu, điều chỉnh dựa trên tình huống trận đấu. Các quyết định của AI được lập trình để tạo thử thách phù hợp với người chơi​.

Các sự kiện môi trường và điều kiện chiến đấu: Hệ thống cập nhật trạng thái của môi trường và các yếu tố tác động như điều kiện thời tiết, thời gian, hoặc khu vực ảnh hưởng đến chiến đấu. Ví dụ, thời gian chờ trong hàng đợi lượt đánh sẽ giúp người chơi biết khi nào đến lượt mình​.

### 2.3.3 Đầu vào từ dữ liệu tài nguyên

Dữ liệu đồ họa và âm thanh: Các tài nguyên như ảnh đồ họa của nhân vật, bản đồ, và hiệu ứng âm thanh, nhạc nền sẽ được load vào trong suốt quá trình chơi. Các tài nguyên này giúp tạo không gian ảo và tăng trải nghiệm của người chơi​​.

Thông tin và thuộc tính của vật phẩm: Dữ liệu về các vật phẩm có thể được thu thập trong game (ví dụ: vũ khí, áo giáp, bình hồi máu) và các thuộc tính tương ứng của chúng sẽ được tải từ kho dữ liệu khi bắt đầu hoặc khi người chơi mở kho đồ​.

### 2.3.4 Đầu vào từ cấu trúc kịch bản và Cutscene

Trigger và sự kiện kịch bản: Khi người chơi di chuyển đến các khu vực đặc biệt hoặc đạt các điều kiện cụ thể, hệ thống sẽ kích hoạt các Cutscene để diễn tiến cốt truyện hoặc mở các nhiệm vụ mới​.

Quản lý trạng thái của Cutscene: Khi một Cutscene bắt đầu, các hành động của người chơi và AI sẽ tạm thời bị dừng để đảm bảo rằng không có sự kiện nào xen vào trong suốt cảnh cắt cảnh, giúp người chơi tập trung vào diễn biến cốt truyện​.

# CHƯƠNG 3: CÀI ĐẶT MÔI TRƯỜNG

### 3.1 Xây dựng thế giới Overworld

- Tạo lớp Gameboard định nghĩa một lưới các ô (dùng kiểu Vector2i) nơi mỗi ô đại diện cho một vị trí của game, xác định ranh giới và kích thước của không gian chơi. Gameboard này được giới hạn bởi các "biên giới" (boundaries) là một hình chữ nhật, và sử dụng kích thước ô (cell size) để tính toán vị trí của các thành phần object trong game như người chơi, npc, obstacles, tile, vật phẩm,…. Người chơi sẽ chỉ có thể di chuyển trong những cell có tọa độ dương và phải nằm trong biên giới.

+ Ô (Cell) đại diện cho một đơn vị trên lưới (grid) của bảng game (gameboard). Bảng game được tạo thành từ các ô xếp theo hàng và cột, và mỗi ô có tọa độ riêng của nó (tọa độ ô). Kích thước mỗi ô sẽ là 16x16 pixel Vector2i(16, 16). Giúp quản lý thế giới trong game theo kiểu lưới, giúp dễ dàng kiểm soát chuyển động, va chạm, và các tương tác khác dựa trên vị trí của các đối tượng trong từng ô.

+ Pixel là đơn vị nhỏ nhất trên màn hình (trong khía cạnh hiển thị). Đây là đơn vị dùng để xác định độ phân giải của game (yếu tố hình ảnh). Trong khi ô (cell) là đơn vị trừu tượng trong lưới game, pixel là đơn vị dùng để hiển thị các ô này trên màn hình. Tọa độ pixel được sử dụng khi bạn cần biết vị trí chính xác của đối tượng trên màn hình. Mỗi ô có thể bao phủ nhiều pixel (trong trường hợp này là 16x16 pixel), nên ta phải chuyển đổi giữa tọa độ ô và tọa độ pixel để vẽ các đối tượng đúng vị trí trên màn hình.

+ Chỉ mục (index) là một con số duy nhất đại diện cho một ô cụ thể trong lưới của bảng game. Thay vì sử dụng hai số (tọa độ x và y) để xác định vị trí của một ô, bạn có thể dùng một chỉ mục duy nhất để chỉ ra ô đó.

Chỉ mục được tính toán bằng cách ánh xạ tọa độ của ô sang một giá trị tuyến tính, giống như cách một mảng 2D có thể được "làm phẳng" thành một mảng 1D.

Giúp đơn giản hóa việc xử lý bài toán tìm đường (pathfinding) ở đây là sẽ tìm đường khi người chơi chọn di chuyển bằng chuột để có thể từ di chuyển từ vị trí hiện tại đến vị trí được click. Mỗi ô có một chỉ mục duy nhất dựa trên vị trí của nó trong lưới.

+ Ta cẩn tọa độ ô giúp định vị các đối tượng trong game theo dạng lưới. Tọa độ pixel giúp bạn hiển thị các đối tượng đó trên màn hình.

Chỉ mục (index) để truy cập hoặc xác định một ô cụ thể trong lưới theo cách tuyến tính và tìm đường. Vậy nên ta sẽ cần tạo các hàm như cell\_to\_pixel(), pixel\_to\_cell(), cell\_to\_index(), lấy những cells liền kề một cell để phục vụ cho việc tìm đường về sau.

- Tạo node Terrain sử dụng node TileMap là lớp cơ bản, chịu trách nhiệm xây dựng các yếu tố địa hình trong game tile shape là các ô vuông (Square), tile size 16x16. Sử dụng một bộ tileset để sắp xếp các ô vuông thành các hình dạng địa hình khác nhau như cỏ, rừng, đường đất, sa mạc,…

A screenshot of a video game

Description automatically generatedA screenshot of a video game

Description automatically generatedA video game screen with a map of a city

Description automatically generated with medium confidenceA screenshot of a video game

Description automatically generated

Hình 3.1: Tileset và ví dụ về cảnh quan, người chơi, npc và item được tạo từ các ô vuông trong trò chơi.

- Về các lớp địa hình: Overworld sẽ có 3 lớp Terrain, Obstacles, Overlay.

+ Terrain(z-index: -1): Lớp chính cho địa hình.

+ Obstacles(z-index: -1): Lớp thuộc tính Collision cho phép xác định các vùng va chạm cho từng tile, đảm bảo rằng người chơi và các đối tượng khác không thể đi qua các khu vực không được phép.

+ Overlay(z-index: -1): Lớp chồng lên nhau, thường dùng cho các chi tiết phụ như cỏ dại hoặc bóng mờ và lá cây.

### 3.2 Xây dựng người chơi và Npc

- Người chơi sẽ có scene là PlayerController (Bộ điều khiển nhân vật người chơi): Đây là thành phần quan trọng cho phép người chơi điều khiển nhân vật trong trò chơi. Mỗi khi nhân vật được người chơi kiểm soát, một instance của PlayerController sẽ được gán cho nhân vật đó. Nhờ đó người chơi có thể  
+ Di chuyển nhân vật

+ Tương tác với môi trường: Khi nhân vật di chuyển đến một vị trí có đối tượng tương tác (Npc, rương vật phẩm), controller sẽ gán đối tượng này làm mục tiêu và nhân vật sẽ thực hiện hành động tương tác khi đến gần đối tượng. Hệ thống này đảm bảo rằng nhân vật chỉ thực hiện tương tác khi đã đến đúng vị trí cần thiết.

+ Quản lý trạng thái di chuyển và dừng: Controller cũng theo dõi và xử lý các sự kiện khi nhân vật đến điểm đích hoặc có thay đổi trong trạng thái đường đi (ví dụ, có chướng ngại vật xuất hiện trên đường di chuyển). Điều này đảm bảo rằng quá trình di chuyển diễn ra mượt mà và logic.

- NPC có thể trò chuyện vơi nhân vật, đưa ra nhiệm vụ, tuần tra, di chuyển theo lộ trình nhất định hoặc tương tác với môi trường. NPC sẽ có scene là AI Controller (Bộ điều khiển AI) NPC được điều khiển bởi một AI controller như PathLoopAIController. Tương tự như người chơi Npc cũng có thể tương tác với môi trường và người chơi(Hiển thị popup khi người chơi đến gần) gọi là Interaction. Ngoài ra còn có thể có trigger ví dụ như combat trigger khiến người chơi vào trạng thái combat.

A pixelated cartoon character

Description automatically generated with medium confidenceA screenshot of a video game

Description automatically generated

A collection of pixel characters

Description automatically generated

Hình 3.2: Sprite sheet của nhân vật, npc biểu tượng trong trò chơi với nhiều tư thế và biểu cảm khác nhau.

### 3.3 Xây dựng nhân vật di chuyển

- Ngườichơi có thể chọn di chuyển bằng 2 cách:

+ Di chuyển bằng bàn phím: Người chơi có thể sử dụng các phím điều hướng để di chuyển nhân vật theo các hướng khác nhau (lên, xuống, trái, phải).

+ Di chuyển bằng chuột: Thao tác với các ô (cell) trên lưới bản đồ (grid). Chức năng chính xác định vị trí của con trỏ (cursor) dựa trên tọa độ chuột, quản lý ô nào được làm nổi bật hoặc được chọn khi người chơi tương tác một marker sẽ được tạo ra tại vị trí đó, chọn vị trí muốn đến bằng thuật toán tìm đường đi để tìm ra tuyến đường tối ưu từ vị trí hiện tại đến điểm đích Thuật toán A\* (A\* for finding the shortest path).

A screenshot of a video game

Description automatically generated

Hình 3.3: Giao diện Godot Engine với sprite nhân vật đang được chỉnh sửa.

### 3.4 Xây dựng camera theo dõi nhân vật

- Mục tiêu:

+ Theo dõi nhân vật chính trong quá trình di chuyển.

+ Đảm bảo không vượt quá giới hạn của bản đồ (gameboard).

+ Tự động điều chỉnh khi kích thước màn hình thay đổi, mang lại trải nghiệm chơi mượt mà và phù hợp trên mọi thiết bị.

- Thiết kế và triển khai:

Camera được thiết kế dựa trên lớp Camera2D của Godot Engine, tận dụng các tính năng mạnh mẽ sẵn có như:

+ Follow (Theo dõi): Camera tự động khóa vào vị trí của nhân vật chính, đảm bảo luôn ở trung tâm khung hình.

+ Boundary (Ranh giới): Giới hạn phạm vi di chuyển của camera trong khu vực bản đồ được định nghĩa trước.

Adaptive Viewport (Khung nhìn thích ứng): Camera sẽ tự động điều chỉnh giới hạn khi kích thước màn hình hoặc tỷ lệ khung hình thay đổi.

- Các bước triển khai:

+ Khởi tạo Camera2D: Thêm node Camera2D làm con của nhân vật chính.

Kích hoạt thuộc tính Current để camera theo dõi nhân vật.

+ Cấu hình ranh giới của camera: Sử dụng thuộc tính Limit trong Camera2D để giới hạn phạm vi:

+ Điều chỉnh khung nhìn theo tỷ lệ màn hình: Lắng nghe sự kiện thay đổi kích thước màn hình (viewport resize) và cập nhật giới hạn camera. Sự kiện này đảm bảo camera không vượt quá ranh giới bản đồ dù màn hình có kích thước lớn hay nhỏ.

+ Kết hợp hiệu ứng mượt: Bật thuộc tính Smoothing để camera di chuyển mượt hơn theo nhân vật

+ Xử lý khi nhân vật gần ranh giới: Camera sẽ không di chuyển ra ngoài ranh giới bản đồ, giữ cho phần khung hình luôn nằm trong phạm vi bản đồ.

### 3.5 Xây dựng Popup

UIPopup là một lớp con của Node2D với các hoạt ảnh cho phép nó xuất hiện, nhảy nhẹ (bounce) và biến mất khi có lệnh. Nó bao gồm các trạng thái và hành vi để quản lý khả năng hiển thị của pop-up, giúp các hiệu ứng trở nên mượt mà và hấp dẫn cho người chơi. Lớp này có thể phát ra tín hiệu disappeared, hữu ích để thông báo cho các phần khác của trò chơi khi pop-up đã hoàn toàn biến mất.

Enum States định nghĩa bốn trạng thái khả dĩ của pop-up:

+ HIDDEN: Pop-up không hiển thị trên màn hình.

+ SHOWN: Pop-up đã hoàn toàn hiển thị và đang ở trạng thái chờ.

+ HIDING: Pop-up đang trong quá trình biến mất.

+ SHOWING: Pop-up đang trong quá trình xuất hiện.

Các trạng thái này giúp lớp quản lý các hoạt ảnh dựa trên trạng thái mục tiêu và trạng thái hiện tại của pop-up.

Biến \_is\_shown là một thuộc tính Boolean đại diện cho việc pop-up nên được hiển thị hay ẩn đi. Khi đặt \_is\_shown, nó sẽ kích hoạt các hoạt ảnh khác nhau dựa trên trạng thái hiện tại:

Nếu \_is\_shown là true và trạng thái là HIDDEN, hoạt ảnh appear sẽ được kích hoạt, thay đổi trạng thái thành SHOWING.

Nếu \_is\_shown là false và trạng thái là SHOWN, pop-up sẽ chuẩn bị để ẩn đi với hoạt ảnh disappear.

Hàm hide\_and\_free cho phép pop-up biến mất và tự động giải phóng khỏi bộ nhớ.

Nếu trạng thái không phải là HIDDEN, \_is\_shown được đặt thành false.

Phương thức này chờ tín hiệu disappeared để đảm bảo pop-up ẩn trước khi gọi queue\_free() để giải phóng nó.

Hàm này quản lý chuyển đổi giữa hoạt ảnh nhảy nhẹ (bounce) và hoạt ảnh ẩn:

Nếu \_is\_shown là true, hàm sẽ phát lại hoạt ảnh bounce\_wait, giúp pop-up thu hút sự chú ý của người chơi một cách nhẹ nhàng.

Nếu \_is\_shown là false, hàm sẽ phát hoạt ảnh disappear, chuyển trạng thái thành HIDING.

### 3.6 Xây dựng hệ thống đối thoại (Dialog)

- Mục tiêu: Xây dựng hệ thống đối thoại cho trò chơi, giúp nhân vật của người chơi có thể tương tác với NPC và các đối tượng trong game thông qua các cuộc hội thoại. Hệ thống này có thể bao gồm các tùy chọn đối thoại, phản hồi của NPC, và các tình huống dẫn dắt câu chuyện trong game.

- Sử dụng plugin của godot Dialogic là một plugin được sử dụng trong Godot Engine để tạo ra các cuộc hội thoại một cách dễ dàng và trực quan. Sử dụng Dialogic, bạn có thể xây dựng hệ thống đối thoại mà không cần phải viết quá nhiều mã phức tạp, nhờ có giao diện người dùng đồ họa trực quan.

- Nhờ đó tạo các đối thoại dễ dàng trực quan cũng như tùy chỉnh hành vi của npc, trigger các sự kiện đặc biệt (Ví dụ như kết thúc trò chơi). Dễ dàng chỉnh sửa , mở rộng và tích hợp

### 3.7 Xây dựng Combat arena system

- Đầu tiên cần một node để gửi tín hiệu (signal). Ta sẽ có 2 signal là combat\_initiated và combat\_finished. combat\_initiated tín hiệu này được phát ra khi trạng thái combat đã được thiết lập xong và sẵn sàng trở thành trạng thái chính trong trò chơi. Tại thời điểm tín hiệu này được phát ra, ta sẽ cần một hiệu ứng chuyển cảnh (transition) để chuyển đổi mượt mà giữa các trạng thái game. combat\_finished tín hiệu này được phát ra khi trạng thái combat đã kết thúc bất kể người chơi có thắng hay thua cho phép các thành phần của trò chơi phản ứng và xử lý các sự kiện hậu combat (ví dụ: phần thưởng, xử lý kết thúc combat, cập nhật trạng thái nhân vật). Ngoài ra ta cần một biến is\_player\_victory: bool lưu trữ thông tin về việc người chơi có chiến thắng trong combat gần nhất hay không nếu như người chơi thua cuộc thì chuyển về Overworld nhưng quái vật vừa tấn công sẽ không biến mất.

- Tiếp đến là class chính CombatArena quản lý âm nhạc khi combat, Turn queue chứa thứ tự lượt hành động của cả người chơi và quái. Turn queue sẽ được quản lý bởi lớp ActiveTurnQueue. Một số signal và biến trạng thái cần thiết:

+ signal combat\_finished(is\_player\_victory: bool): Được phát ra khi trận chiến kết thúc. Nó truyền một biến boolean để cho biết liệu người chơi có chiến thắng hay không.

+ signal player\_turn\_finished: Được phát ra khi một lượt đánh của người chơi kết thúc, để thông báo cho hệ thống rằng người chơi đã hoàn thành hành động của mình.

+ signal skill\_selected(action: BattlerAction) và signal target\_selected(targets: Array[Battler]): Được phát ra khi người chơi chọn kỹ năng hoặc mục tiêu trong lượt của mình.

+ signal turn\_queue\_ready: Tín hiệu này được phát ra khi hàng đợi lượt đánh đã sẵn sàng, báo hiệu rằng mọi thứ đã được thiết lập để bắt đầu trận chiến.

+ is\_active: Kiểm soát xem hệ thống có đang hoạt động hay không. Khi trận chiến đang diễn ra, is\_active được đặt thành true, và khi kết thúc trận chiến hoặc trong một số tình huống nhất định (chuyển cảnh), nó sẽ được đặt thành false.

+ time\_scale: Hệ số thời gian điều khiển tốc độ của trận chiến. Điều này cho phép làm chậm thời gian khi người chơi đang chọn hành động time\_scale = 0 chờ input skill(action) và target từ người chơi.

+ \_queued\_player\_battlers và \_combined\_turn\_queue: Đây là các hàng đợi chứa các nhân vật đang chờ đến lượt hành động. \_queued\_player\_battlers chứa các nhân vật thuộc người chơi, còn \_combined\_turn\_queue chứa cả nhân vật của người chơi và đối thủ để tạo UI hàng chờ giúp người chơi biết khi nào đến lượt.

+ Tiếp đến ta sẽ cần khởi tạo các indicators để người chơi chọn mục tiêu mà skill sẽ tác động như gây dame, hồi máu, buff. Khi người chơi chọn mục tiêu thành công sẽ lưu vào biến target, còn skill khi được chọn đc lưu vào action[index] index là skill được thực hiện. Còn về phí kẻ địch những target và action sẽ được chọn random.

+ Xử lý khi nhân vật bị hạ gục (health\_depleted): Khi một nhân vật bị hạ gục (mất hết máu), chỉ báo của họ sẽ bị xóa khỏi màn hình, và họ sẽ bị xóa khỏi hàng đợi lượt đánh. Hệ thống kiểm tra xem toàn bộ đội hình của người chơi hoặc đối thủ có bị hạ gục hay không. Nếu có, trận chiến sẽ kết thúc và phát ra tín hiệu combat\_finished.

+ Khi tất cả các hoạt ảnh của nhân vật đã hoàn thành, hệ thống sẽ phát ra tín hiệu combat\_finished, kết thúc trận chiến và trả lại quyền kiểm soát cho hệ thống quản lý trò chơi chung.

A screenshot of a computer

Description automatically generated

Hình 3.4: Scene combat với bố cục và thuộc tính của các node trong cảnh.

A screenshot of a video game

Description automatically generated

Hình 3.5: Ví dụ một instance của combat scene

### 3.8 Xây dụng lớp Battler

Lớp Battler chịu trách nhiệm đại diện cho các nhân vật tham gia trận chiến, bao gồm cả người chơi và kẻ địch. Battler thực hiện các hành động chiến đấu, tương tác với hệ thống BattlerStats và BattlerActions. Dưới đây là một số chi tiết và ví dụ về cách triển khai lớp Battler trong trò chơi. Trong hệ thống chiến đấu theo lượt, mỗi Battler có một lượt hành động và khi đến lượt, hệ thống sẽ kiểm tra trạng thái của Battler để quyết định hành động tiếp theo.

A screenshot of a computer

Description automatically generated

Hình 3.6: Định nghĩa lớp Battler với các tín hiệu và thuộc tính.

A screenshot of a computer program

Description automatically generated

Hình 3.7: Ví dụ instace của lớp BattlerStats chứa thông tin của Battler.

Lớp Battler này chứa các phương thức để điều khiển trạng thái của nhân vật, bao gồm việc tấn công, nhận sát thương, và kiểm tra khi nào nhân vật sẵn sàng để hành động. Battler cũng có hệ thống tín hiệu như health\_updated để cập nhật HP, signal energy\_updated(value: int) MP cho nhân vật mỗi khi kỹ năng actions yêu cầu năng lượng > 0.

Quyết định tốc độ nạp lượt (readiness) để nhân vật có thể thực hiện hành động tiếp theo. Tốc độ nhân vật (speed) ở lớp BattlerStats sẽ ảnh hưởng đến việc nhân vật có thể hành động nhanh hay chậm. Khi người chơi đạt readiness = 100 đồng nghĩa với việc người chơi hoặc kẻ địch (được xác định nhờ biến is\_player) đó thực hiện kỹ năng của mình.

A screen shot of a computer code

Description automatically generated

Hình 3.8: Thuộc tính readiness của Battler giúp hệ thống xác định lượt.

@export var icon: Texture sẽ lưu trữ icon để hiện thị trên Turnqueue UI giúp người chơi biết khi nào sẽ tới lượt của mình giúp xây dựng chiến thuật nên thực hiện hành động gì tiếp theo.

A screenshot of a video game

Description automatically generated

Hình 3.9: Giao diện của TurnQueue giúp người chơi xác định thời điểm tới lượt.

Khi nhân vật thực hiện một hành động tấn công, hệ thống sẽ trừ đi năng lượng và thực hiện hành động tương ứng với đối thủ. Sẽ kết thực lượt hành động và pop() người chơi đó khỏi TurnQueue để nhường lượt chó người chơi kế tiếp

A screen shot of a computer program

Description automatically generated

Hình 3.10: Đoạn mã GDScript thực hiện hành động tấn công của nhân vật bao gồm việc trừ năng lượng và xử lý lượt chơi.

A screenshot of a video game

Description automatically generated

Hình 3.11: Tổng quan giao diện combat

Khi người chơi bị chọn làm mục tiêu từ kĩ năng của kẻ địch hàm take\_hit của instance người chơi đó sẽ được thực thi. Việc hp của người chơi sẽ bị tiêu hao hay không phụ thuộc vào đòn đánh có thành công hay không hit.is\_successful(). Hit.is\_successful() sẽ phụ thuộc vào nhưng thuộc tính như Evasion của bản thân người chơi bị chọn làm mục tiêu và hit\_chance của kẻ địch thực hiện kĩ năng.

Vậy nên để evasion sẽ giúp Battler đó tăng khả năng né chiêu còn hit\_chance sẽ tăng khả năng người bị ảnh hưởng bởi kĩ năng nhận sát thương. Do đó ngoài những thuộc tính như hp, attack, speed,… thì hai thuộc tính evasion và hit\_chance cũng rất quan trọng trong combat.

A screen shot of a computer code

Description automatically generated



Hình 3.12: Giao diện, Logic xử lý khi Battler chịu ảnh hưởng bởi action

### 3.9 Xây dựng Stats cho các Battler

Mỗi Battler có các chỉ số (stats) riêng, đại diện cho khả năng của nhân vật trong chiến đấu. Các chỉ số này bao gồm sức khỏe (HP), sức mạnh tấn công (Attack), khả năng phòng thủ (Defense), và tốc độ (Speed). Khi nhân vật lên cấp hoặc trang bị các vật phẩm, các chỉ số này sẽ được tăng lên tương ứng, ảnh hưởng trực tiếp đến khả năng sống sót và hiệu quả chiến đấu của nhân vật​

Lớp **BattlerStats** đại diện cho các chỉ số của một **Battler** cụ thể, bao gồm sức khỏe, năng lượng, sức mạnh tấn công, phòng thủ, và nhiều chỉ số khác. Những chỉ số này xác định khả năng chiến đấu của **Battler** trong trò chơi. Bên cạnh đó, lớp này cũng hỗ trợ các thuộc tính có thể được điều chỉnh bằng các **modifier** (cộng/trừ giá trị) hoặc **multiplier** (nhân giá trị), giúp cho hệ thống chỉ số linh hoạt và dễ dàng tùy chỉnh.

**Các chỉ số chính:**

1. **max\_health**: Sức khỏe tối đa của nhân vật.
2. **max\_energy**: Năng lượng tối đa mà nhân vật có thể sử dụng cho các hành động đặc biệt.
3. **attack**: Chỉ số tấn công quyết định sát thương mà nhân vật gây ra khi tấn công.
4. **defense**: Chỉ số phòng thủ quyết định khả năng giảm sát thương khi bị tấn công.
5. **speed**: Tốc độ của nhân vật, ảnh hưởng đến thời gian để đến lượt hành động.
6. **hit\_chance**: Xác suất trúng đòn khi nhân vật tấn công.
7. **evasion**: Khả năng né đòn của nhân vật khi bị tấn công.

**MODIFIABLE\_STATS**: Danh sách các chỉ số có thể nhận được các giá trị modifier để điều chỉnh.

**Tín hiệu (signal**): Khi chỉ số sức khỏe hoặc năng lượng thay đổi, tín hiệu sẽ được phát để cập nhật giao diện người dùng như thanh máu (HP bar) hoặc thanh năng lượng (Mana bar).

**health\_depleted**: Phát ra khi máu của nhân vật về 0.

**health\_changed**: Phát ra mỗi khi máu thay đổi.

**energy\_changed**: Phát ra khi năng lượng của nhân vật thay đổi.

Tính toán và giới hạn giá trị:

**clampi(value, 0, max\_health)**: Hàm này đảm bảo rằng máu không thể nhỏ hơn 0 hoặc lớn hơn giá trị tối đa đã định (max\_health).

energy: Tương tự như máu, năng lượng cũng được giới hạn trong khoảng từ 0 đến max\_energy.

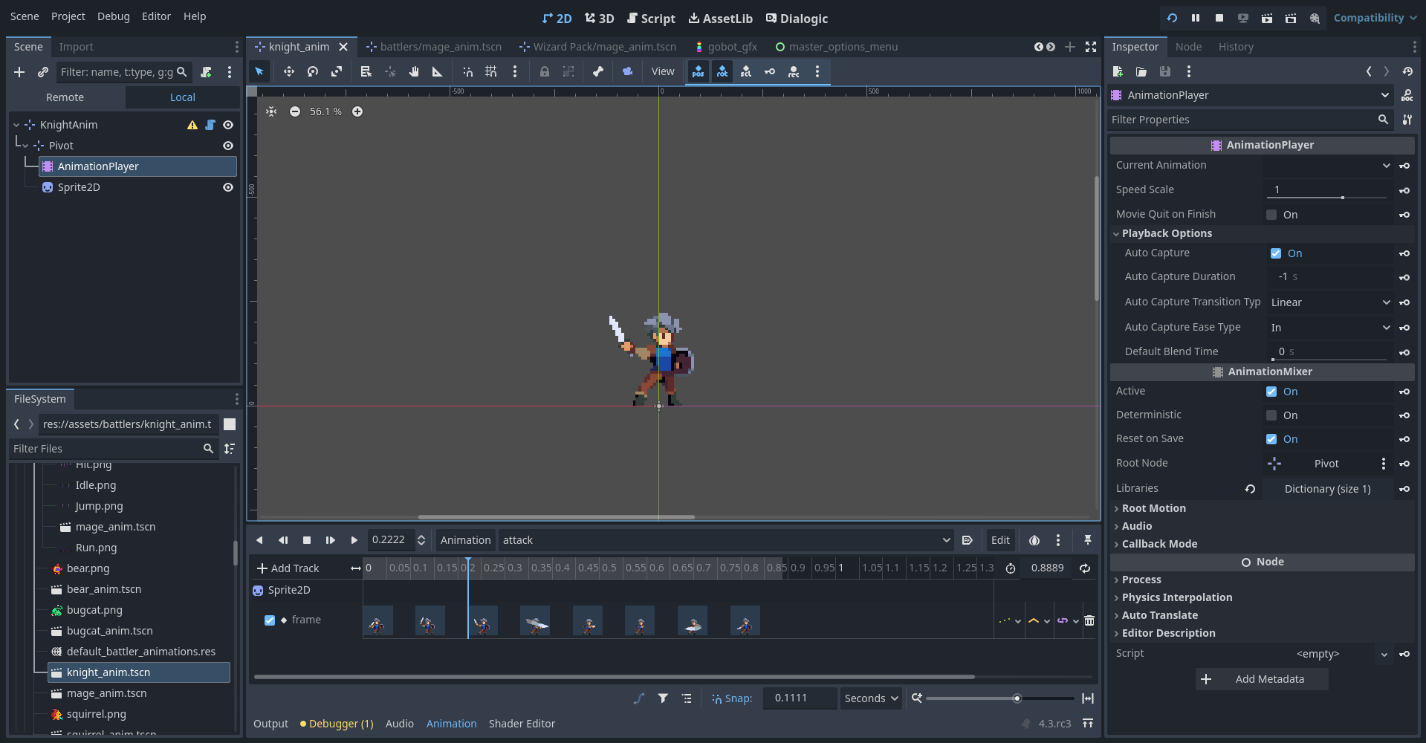
Lớp BattlerStats còn cho phép thêm các modifier và multiplier để điều chỉnh chỉ số của nhân vật. Modifier là các giá trị được cộng hoặc trừ trực tiếp vào chỉ số, trong khi Multiplier là các hệ số nhân giúp tăng giảm theo tỷ lệ phần trăm. **Modifier** và **multiplier** trong hệ thống của **BattlerStats** được thiết kế để hỗ trợ cho các kỹ năng **buff** và **debuff**. Đây là cách hiệu quả để điều chỉnh các chỉ số của nhân vật tạm thời hoặc lâu dài thông qua các trạng thái (status effects) mà kỹ năng tạo ra.

=> BattlerStats cung cấp một cơ chế mạnh mẽ và linh hoạt để quản lý các chỉ số của nhân vật trong trò chơi. Nó cho phép điều chỉnh chỉ số theo thời gian thực thông qua việc thêm hoặc xóa các modifier và multiplier, từ đó giúp hệ thống chiến đấu trở nên phong phú và phức tạp hơn.

Bạn có thể mở rộng thêm lớp này để bao gồm các chỉ số hoặc tín hiệu khác nếu cần, đồng thời điều chỉnh các phương thức tính toán để phù hợp với các yêu cầu cụ thể của trò chơi.

### 3.10 Xây dựng Animation cho các Battler

Hoạt ảnh (animation) là một phần quan trọng trong việc tạo cảm giác chân thực cho trận chiến. Battler được gán các hoạt ảnh như di chuyển, tấn công, sử dụng kỹ năng và bị thương. Hệ thống hoạt ảnh sử dụng AnimationPlayer và AnimationTree, giúp dễ dàng điều chỉnh và đồng bộ các hoạt ảnh với hành động của Battler. Khi một hành động được kích hoạt, tương ứng hoạt ảnh sẽ được phát



Hình 3.13: Node BattlerAnim chứa sprite và animation của nhân vật.

### 3.11 Xây dựng Battler action

Battler Action đại diện cho các hành động mà Battler có thể thực hiện trong trận chiến. Các hành động bao gồm tấn công, sử dụng kỹ năng, hoặc phòng thủ. Khi đến lượt của nhân vật, hệ thống sẽ cung cấp cho người chơi các tùy chọn về hành động có thể thực hiện, và sau khi hành động được chọn, hệ thống sẽ xử lý nó tương ứng

A screenshot of a computer

Description automatically generated

Figure 3.14: Giao diện chọn action (skill) và một số thuộc tính cơ bản action.

Lớp **BattlerAction** đại diện cho các hành động mà một **Battler** có thể thực hiện trong lượt của mình. Đó có thể là các hành động tấn công, phòng thủ, hoặc sử dụng các kỹ năng đặc biệt. Lớp này đóng vai trò là một interface, cho phép các hành động cụ thể kế thừa và triển khai phương thức execute. Phương thức này chứa logic để thực hiện hành động bao gồm các hiệu ứng hoặc hoạt ảnh.

* **icon**: Biểu tượng của hành động, được hiển thị trong các menu để người chơi có thể chọn.
* **label**: Tên của hành động, hiển thị trong menu hoặc UI.
* **description**: Mô tả chi tiết về hành động, giúp người chơi hiểu rõ hơn về tác dụng của nó.
* **energy\_cost**: Chi phí năng lượng (Mana hoặc một loại tài nguyên khác) mà hành động cần để thực hiện.
* **element**: Yếu tố (Element) mà hành động liên quan tới, có thể là lửa, nước, đất, hoặc không liên quan đến bất kỳ yếu tố nào.
* **targets\_self và targets\_all**: Quy định xem hành động có tự động nhắm vào chính mình hoặc tất cả các mục tiêu trên chiến trường hay không.
* **readiness\_saved**: Chỉ số cho biết lượng sẵn sàng còn lại sau khi hành động được thực hiện. Giá trị này có thể được sử dụng để thiết kế các hành động yếu nhưng giúp Battler hành động nhanh hơn ở lượt tiếp theo.

Phương thức **can\_be\_used\_by**(battler) sẽ kiểm tra liệu nhân vật có đủ năng lượng để thực hiện hành động hay không. Phương thức execute sẽ chứa logic thực thi hành động, đây là phương thức quan trọng và phải được ghi đè (override) bởi các lớp con để thực hiện các hành động cụ thể.

Các kỹ năng trong tương lai sẽ kế thừa từ lớp BattlerAction này để đa dạng hơn mỗi Battler sẽ có những kỹ năng khác nhau. Vì vậy người chơi cần chọn kỹ năng để phòng thủ và tấn công sao cho hợp lý để dành chiến thắng.

A screenshot of a computer program

Description automatically generated

Hình 3.15: Ví dụ một kỹ năng được kế thừa (extend) từ BattlerAction

### 3.12 Xây dụng Quest

Hệ thống nhiệm vụ (Quest) cho phép người chơi tham gia vào các hoạt động và nhiệm vụ đa dạng trong trò chơi. Mỗi nhiệm vụ có các mục tiêu như tiêu diệt quái vật, thu thập vật phẩm hoặc tìm kiếm NPC. Khi hoàn thành nhiệm vụ, người chơi sẽ nhận được phần thưởng như vật phẩm hoặc kinh nghiệm để tăng cấp cho nhân vật

A video game screen with a pixelated village

Description automatically generated

Hình 3.16: Hình ảnh tương tác với npc để nhận quest.

### 3.13 Xây dựng chức năng cài đặt (Setting)

- Menu cài đặt gồm:  
 + Scene điều chỉnh âm thanh: Bao gồm các thanh trượt (sliders) để người chơi có thể điều chỉnh âm lượng tổng thể, âm lượng nhạc nền (background music), và âm lượng hiệu ứng âm thanh (sound effects).

A screenshot of a computer

Description automatically generated

Hình 3.17: Giao diện cài đặt âm thanh.

+ Scene cài đặt đồ họa:

Các tùy chọn để người chơi có thể điều chỉnh chất lượng đồ họa như độ phân giải màn hình, bật/tắt hiệu ứng đổ bóng, độ phân giải texture, và tỷ lệ khung hình.

A screenshot of a computer

Description automatically generated

Hình 3.18: Giao diện cài đặt đồ họa.

### 3.14 Xây dựng Interaction

Lớp Interaction kế thừa từ Cutscene và được sử dụng để quản lý các tương tác giữa người chơi và các đối tượng trong game, như các điểm lưu game, rương báu, hoặc cuộc hội thoại với NPC. Interaction dựa trên việc người chơi di chuyển đến gần một khu vực tương tác (thường được phát hiện bởi CollisionShape2D) và thực hiện một hành động, ví dụ như nhấn phím Spacebar để kích hoạt tương tác.

Các thuộc tính và tín hiệu chính:

**is\_active**: Đây là thuộc tính quyết định liệu Interaction có thể được kích hoạt hay không. Khi nó được đặt thành false, Interaction sẽ không phản hồi các sự kiện của người chơi, trừ khi được kích hoạt qua code trực tiếp bằng phương thức run().

**\_overlapping\_areas**: Danh sách các khu vực (Area2D) đang nằm trong vùng tương tác của Interaction, tức là các vùng va chạm mà người chơi đang ở trong. Nó giúp kiểm tra xem người chơi có đang trong phạm vi có thể tương tác hay không.

**Phương thức hoạt động:**

Kích hoạt tương tác qua phím bấm:

Khi người chơi đến gần một Interaction và nhấn phím tương tác (ví dụ: Spacebar), hệ thống sẽ kích hoạt cắt cảnh hoặc sự kiện tương ứng bằng cách gọi phương thức run().

A screen shot of a computer

Description automatically generated

Hình 3.19: Mã xử lý khi người chơi tương tác.

- Xác định trạng thái có thể tương tác: Phương thức \_update\_input\_state kiểm tra xem Interaction có thể hoạt động hay không dựa trên việc có đối tượng (người chơi) đang trong phạm vi tương tác và Interaction đang ở trạng thái active.

Nếu có ít nhất một Area2D (người chơi) nằm trong vùng tương tác và Interaction đang active, trạng thái của Interaction sẽ được chuyển sang cho phép xử lý input. Khi đó, người chơi có thể kích hoạt tương tác bằng cách nhấn phím tương ứng.

A black background with white text

Description automatically generated

Hình 3.20: Phương thức \_update\_input\_state trong mã nguồn kiểm tra trạng thái có thể tương tác của một đối tượng.

- Phát hiện người chơi đi vào vùng tương tác: Khi người chơi di chuyển vào vùng va chạm của Interaction, phương thức \_on\_area\_entered được gọi và đối tượng tương ứng được thêm vào danh sách \_overlapping\_areas.

Phương thức này đảm bảo rằng các tương tác chỉ được kích hoạt khi người chơi đang ở trong vùng tương tác và phương thức \_update\_input\_state sẽ được gọi để cập nhật trạng thái của Interaction.

A black screen with white text

Description automatically generated

Hình 3.21: Phương thức \_on\_area\_entered trong mã nguồn được gọi khi người chơi di chuyển vào vùng va chạm của Interaction.

- Xử lý khi người chơi rời khỏi vùng tương tác: Nếu người chơi rời khỏi vùng tương tác, phương thức \_on\_area\_exited sẽ xóa khu vực tương ứng khỏi danh sách \_overlapping\_areas, đồng thời cập nhật lại trạng thái có thể tương tác.

A black screen with white text

Description automatically generated

Hình 3.22: Phương thức \_on\_area\_exited trong mã nguồn được gọi khi người chơi rời khỏi vùng tương tác.

- Tạm dừng va chạm: Khi trò chơi tạm dừng, các va chạm và tương tác cũng có thể bị tạm dừng, đảm bảo rằng không có sự kiện nào xảy ra khi trò chơi đang dừng. Phương thức \_on\_input\_paused đảm bảo rằng các đối tượng va chạm không còn gửi tín hiệu khi trò chơi đang bị tạm dừng.

A computer screen with white text

Description automatically generated

Hình 3.23: Phương thức \_on\_input\_paused trong mã nguồn được sử dụng để tạm dừng các va chạm và tương tác khi trò chơi bị tạm dừng.

- Cảnh báo cấu hình: Phương thức \_get\_configuration\_warnings kiểm tra xem có đối tượng nào được kết nối với tín hiệu va chạm của Interaction hay không. Nếu không có kết nối, nó sẽ phát ra cảnh báo rằng Interaction sẽ không thể hoạt động được.

### 3.15 Xây dựng Trigger

Lớp Trigger trong trò chơi chịu trách nhiệm phát hiện khi một Gamepiece (nhân vật hoặc đối tượng trong game) đi vào khu vực của nó và kích hoạt các sự kiện tương ứng. Lớp này kế thừa từ Cutscene, có nghĩa là nó không chỉ dừng lại ở việc phát hiện va chạm mà còn có thể bắt đầu một đoạn cảnh cắt cảnh (cutscene) khi kích hoạt.

Các Trigger có thể được thiết kế để hoạt động trong nhiều tình huống khác nhau, chẳng hạn như mở một cánh cửa khi người chơi đi qua, kích hoạt một nhiệm vụ mới, hoặc kích hoạt một đoạn hội thoại.

Các thuộc tính và tín hiệu chính:

**is\_active**: Đây là thuộc tính để kích hoạt hoặc vô hiệu hóa Trigger. Nếu Trigger không hoạt động, nó sẽ không kích hoạt các sự kiện ngay cả khi có đối tượng va chạm.

**Tín hiệu (Signal):**

**gamepiece\_entered:** Được phát ra khi một Gamepiece bắt đầu di chuyển đến vị trí của Trigger.

**gamepiece\_exited**: Được phát ra khi Gamepiece rời khỏi vị trí của Trigger.

triggered: Được phát ra khi Gamepiece hoàn thành việc di chuyển vào vị trí và kích hoạt Trigger.

A video game screen with a cartoon character

Description automatically generated

Hình 3.24: Ví dụ 1 combat trigger.

### 3.16 Xây dựng Equipment

Hệ thống Equipment cho phép người chơi trang bị các vật phẩm như vũ khí, áo giáp, và phụ kiện để tăng cường sức mạnh của nhân vật. Khi một vật phẩm được trang bị, các chỉ số của nhân vật như sức tấn công, phòng thủ sẽ được tăng cường. Vật phẩm cũng có thể thay đổi diện mạo của nhân vật trong trò chơi.

Tạo lớp Equipment kế thừa từ CanvasLayer và chịu trách nhiệm quản lý giao diện trang bị và chỉ số của nhân vật trong trò chơi. Nó cho phép người chơi tương tác với các vật phẩm trang bị như vũ khí, áo giáp, mũ, và phụ kiện, đồng thời cập nhật các chỉ số tương ứng dựa trên trang bị và nhân vật hiện tại. Các chỉ số của nhân vật bao gồm HP, Energy, Attack, Defense, Speed, Hit Chance, và Evasion.

A screenshot of a video game

Description automatically generated

Hình 3.25: Giao diện trang bị và thông số nhân vật

+ weapon\_slot, armor\_slot, head\_slot, accessory\_slot: Đây là các ô giao diện (slot) đại diện cho các trang bị khác nhau mà nhân vật có thể sử dụng.

equipment\_inventory\_ui: Khu vực giao diện chứa các vật phẩm trang bị mà người chơi có thể chọn để sử dụng.

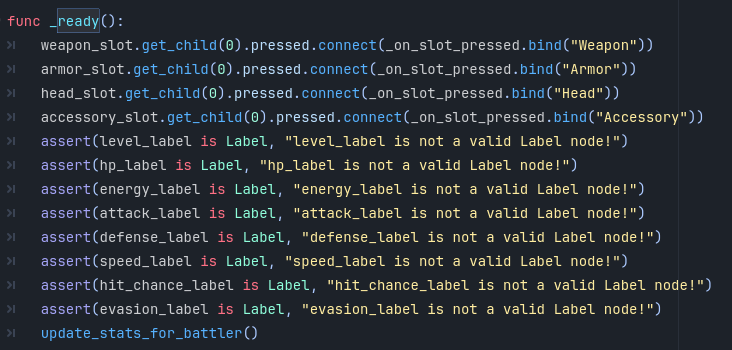
+ level\_label, hp\_label, energy\_label, attack\_label, defense\_label, speed\_label, hit\_chance\_label, evasion\_label: Các nhãn hiển thị thông tin về chỉ số của nhân vật như cấp độ, máu, năng lượng, và các chỉ số khác.

+ equipped\_items: Một từ điển lưu trữ các vật phẩm trang bị hiện tại của nhân vật trong các vị trí như vũ khí, áo giáp, mũ và phụ kiện.

current\_battler: Đại diện cho nhân vật hiện tại (mặc định là "Knight").

current\_stats: Lưu trữ các chỉ số hiện tại của nhân vật.

- \_ready(): Phương thức này kết nối sự kiện nhấn vào các ô trang bị với các phương thức xử lý tương ứng. Nó cũng kiểm tra sự tồn tại và tính hợp lệ của các nhãn hiển thị chỉ số, đồng thời cập nhật chỉ số nhân vật khi khởi động.



Hình 3.26: Phương thức \_ready() trong mã nguồn kết nối các sự kiện nhấn vào các ô trang bị với các phương thức xử lý tương ứng, bao gồm vũ khí, áo giáp, mũ và phụ kiện.

- \_on\_slot\_pressed(slot\_type):

Phương thức này được gọi khi người chơi nhấn vào một trong các ô trang bị (vũ khí, áo giáp, mũ, phụ kiện). Nó sẽ hiển thị kho đồ liên quan đến loại trang bị đó.

A computer screen shot of a computer code

Description automatically generated

Hình 3.27: Phương thức \_on\_slot\_pressed(slot\_type) trong mã nguồn, được gọi khi người chơi nhấn vào ô trang bị, nhằm hiển thị kho đồ tương ứng với loại trang bị (vũ khí, áo giáp, mũ, phụ kiện).

- show\_equipment\_inventory(slot\_type): Phương thức này hiển thị danh sách các vật phẩm trang bị có thể được sử dụng trong ô đã chọn. Nó lọc danh sách vật phẩm dựa trên loại trang bị và cập nhật giao diện tương ứng.

- equip\_item(item, slot\_type):

Phương thức này trang bị một vật phẩm vào ô đã chọn và cập nhật chỉ số của nhân vật. Nó sẽ gán vật phẩm vào ô tương ứng và áp dụng các chỉ số cộng thêm từ vật phẩm đó.

A screenshot of a computer screen

Description automatically generated

Hình 3.28: Phương thức thêm trang bị cho nhân vật.

- unequip\_item(slot\_type):

Phương thức này gỡ bỏ vật phẩm hiện tại trong một ô trang bị và trả lại các chỉ số gốc cho nhân vật. Nó cũng cập nhật lại giao diện và chỉ số sau khi gỡ bỏ.

A computer screen shot of a code

Description automatically generated

Hình 3.29: Phương thức xử lý chức năng tháo trang bị của nhân vật.

- update\_stats\_for\_battler():

Phương thức này cập nhật giao diện chỉ số của nhân vật dựa trên thông tin từ GlobalData (dữ liệu người chơi) và current\_stats (chỉ số hiện tại của nhân vật).

A screen shot of a computer program

Description automatically generated

Hình 3.30: Phương thức cập nhật thông số của nhân vật trên giao diện

- \_on\_item\_pressed(item\_type):

Khi người chơi chọn một vật phẩm từ kho đồ, phương thức này sẽ xử lý sự kiện nhấn và quyết định trang bị hoặc gỡ bỏ vật phẩm đó.

A screen shot of a computer program

Description automatically generated

Hình 3.31: Phương thức xử lý đầu vào từ chuột khi click vào giao diện.

- \_on\_item\_hover(item\_type):

Khi người chơi di chuột qua một vật phẩm trong kho đồ, phương thức này sẽ hiển thị sự khác biệt về chỉ số giữa vật phẩm được trang bị hiện tại và vật phẩm đang di chuột qua.

A screen shot of a computer program

Description automatically generated

Hình 3.32: Phương thức \_on\_item\_hover(item\_type) trong mã nguồn, được gọi khi người chơi di chuột qua một vật phẩm trong kho đồ.

Lớp này cung cấp một giao diện hoàn chỉnh để quản lý trang bị và chỉ số của nhân vật trong trò chơi. Người chơi có thể dễ dàng thay đổi trang bị, xem các chỉ số tương ứng, và quyết định xem vật phẩm nào phù hợp nhất cho chiến đấu. Việc so sánh trực tiếp giữa các vật phẩm cũng giúp người chơi có cái nhìn rõ ràng hơn về lợi ích của từng vật phẩm.

### 3.17 Xây dựng hệ thống Shop

- Hệ thống này cho phép người chơi mua các vật phẩm từ cửa hàng sử dụng số lượng coin (tiền) mà họ sở hữu. Dưới đây là mô tả chi tiết về các thành phần và chức năng trong mã:

Các thành phần chính:

items: items là một danh sách chứa thông tin về các vật phẩm có sẵn trong cửa hàng. Mỗi vật phẩm bao gồm tên, giá, hình ảnh, và loại vật phẩm (item\_type). Ví dụ: "RedWand", "Bomb", và "Key" là các vật phẩm có thể được mua trong cửa hàng.

- @onready var anim\_player: anim\_player là một AnimationPlayer được sử dụng để chơi các hiệu ứng chuyển tiếp (transition) khi mở hoặc đóng cửa hàng.

- @onready var grid\_container: grid\_container là một GridContainer nằm bên trong một ScrollContainer dùng để hiển thị danh sách các vật phẩm mà người chơi có thể mua.

- @onready var player\_gold\_label: player\_gold\_label là một Label hiển thị số lượng coin mà người chơi hiện đang sở hữu.

- @onready var message\_label: message\_label là một Label dùng để hiển thị thông điệp tới người chơi, ví dụ như khi người chơi không đủ tiền mua hoặc khi mua thành công một vật phẩm.

A screenshot of a video game

Description automatically generated

Hình 3.33: Tương tác với npc Smith để mở chức năng shop.

Các phương thức chính:

- \_ready(): Phương thức này được gọi khi đối tượng xuất hiện lần đầu tiên trong cây cảnh (scene tree). Nó kết nối tín hiệu Dialogic.signal\_event để xử lý sự kiện khi người chơi mở cửa hàng. Đồng thời, nó gọi phương thức update\_shop\_ui() để cập nhật giao diện cửa hàng dựa trên số lượng tiền hiện có và các vật phẩm có sẵn trong kho.

- play\_trans\_in() và play\_trans\_out(): Hai phương thức này kiểm soát việc mở và đóng cửa hàng bằng cách chơi các hoạt cảnh chuyển tiếp (transition) thông qua AnimationPlayer. Khi play\_trans\_in() được gọi, cửa hàng sẽ mở và ngược lại với play\_trans\_out() khi đóng.

- update\_shop\_ui(): Phương thức này cập nhật giao diện cửa hàng:

+ Xóa các vật phẩm cũ khỏi GridContainer.

+ Hiển thị số lượng coin hiện có của người chơi.

+ Tạo ra các phần tử giao diện (như hình ảnh, tên, giá, và nút "Buy") cho mỗi vật phẩm có sẵn trong cửa hàng.

+ Kết nối nút "Buy" với phương thức \_on\_buy\_button\_pressed() để xử lý khi người chơi nhấn vào nút mua.

- \_on\_buy\_button\_pressed(item): Phương thức này được gọi khi người chơi nhấn nút mua một vật phẩm. Kiểm tra xem người chơi có đủ tiền mua vật phẩm hay không.

Nếu người chơi đủ tiền, số coin sẽ bị trừ đi và vật phẩm sẽ được thêm vào kho của người chơi. Thông điệp thành công sẽ được hiển thị trên message\_label.

Nếu không đủ tiền, sẽ hiển thị thông báo "Not enough coins to buy [item\_name]."

- DialogicSignal(argument: String): Phương thức này kết nối với tín hiệu từ hệ thống Dialogic. Khi nhận được sự kiện "open\_shop", nó sẽ mở cửa hàng nếu cửa hàng chưa mở.

A screenshot of a video game

Description automatically generated

Hình 3.34: Giao diện chức năng shop chứa những item cần thiết cho người chơi.

### 3.18 Xây dựng Cutscene

Lớp Cutscene trong trò chơi đại diện cho một sự kiện ngắn có kịch bản sẵn, tạm dừng gameplay để thực hiện các hoạt động được lập trình trước, tương tự như một cảnh quay trong phim. Ví dụ, Cutscene có thể hiển thị cuộc hội thoại giữa các nhân vật, chuyển cảnh để hiển thị một sự kiện quan trọng, hoặc thay đổi các yếu tố trong trò chơi như kho đồ của người chơi. Trong suốt thời gian diễn ra Cutscene, tất cả các hoạt động khác trên game trường (field gameplay) sẽ tạm ngừng cho đến khi Cutscene kết thúc.

- Dừng gameplay: Khi một Cutscene bắt đầu, nó sẽ phát tín hiệu tạm dừng FieldEvents.input\_paused để thông báo cho tất cả các đối tượng khác trên game trường, bao gồm cả AI và các đối tượng điều khiển của người chơi, dừng lại. Tất cả các hành động của người chơi và AI sẽ bị vô hiệu hóa cho đến khi Cutscene kết thúc. Tính năng này đảm bảo rằng không có hành động nào của người chơi hoặc AI có thể tiếp tục trong khi Cutscene đang diễn ra, cho phép tập trung hoàn toàn vào sự kiện trong Cutscene.

A screen shot of a computer program

Description automatically generated

Hình 3.35:Đoạn mã triển khai trạng thái is\_cutscene\_in\_progress, sử dụng một thuộc tính tĩnh (static variable) để theo dõi liệu một cảnh cắt (cutscene) có đang diễn ra trong game hay không.

- Lớp Cutscene được thiết kế để đảm bảo rằng chỉ có thể có một Cutscene hoạt động trong một thời điểm nhất định. Điều này được kiểm soát bởi biến \_is\_cutscene\_in\_progress, biến này sẽ theo dõi trạng thái của Cutscene. Nếu đã có một Cutscene đang chạy, các Cutscene khác sẽ không được bắt đầu cho đến khi Cutscene hiện tại kết thúc.

- Tạm dừng và tiếp tục: Khi một Cutscene kết thúc, trạng thái paused sẽ được giải phóng và gameplay sẽ tiếp tục như bình thường. Phương thức run() bắt đầu Cutscene, sau đó đợi cho đến khi phương thức \_execute() (được ghi đè bởi các lớp con) hoàn thành, và sau đó thiết lập lại trạng thái gameplay.

- Phương thức \_execute() là nơi mà các sự kiện cụ thể của Cutscene sẽ được thực hiện. Mỗi Cutscene tùy chỉnh sẽ ghi đè phương thức này để triển khai các hành động cụ thể, chẳng hạn như hiển thị hội thoại, thay đổi môi trường, hoặc thay đổi trạng thái của nhân vật. Phương thức này có thể là không đồng bộ (asynchronous), nghĩa là có thể đợi các hành động xảy ra trước khi tiếp tục.

### 3.19 Xây dựng hệ thống Inventory

Lớp Inventory: Lớp Inventory kế thừa từ Resource, cho phép lưu và tải kho đồ dưới dạng tệp tài nguyên trong Godot.

Các loại vật phẩm: enum ItemTypes định nghĩa các loại vật phẩm khác nhau (ví dụ: chìa khóa, tiền xu, bình thuốc, và vũ khí) mà hệ thống kho đồ sẽ theo dõi.

Biểu tượng: Từ điển ICONS liên kết mỗi loại vật phẩm với một biểu tượng, được tải từ tệp hình ảnh (atlas texture). Các biểu tượng này được preload và gắn kết với vật phẩm tương ứng, giúp dễ dàng hiển thị.

\_init(): Hàm khởi tạo Inventory tạo ra một từ điển rỗng \_items để lưu trữ số lượng vật phẩm, mỗi loại vật phẩm được khởi tạo với giá trị bằng 0.

restore(): Hàm tĩnh này phục hồi kho đồ từ tệp lưu (inventory.tres). Nếu tệp không tồn tại, nó tạo một kho đồ mới.

add(): Thêm một loại vật phẩm vào kho, đồng thời kiểm tra để đảm bảo số lượng vật phẩm không âm.

remove(): Giảm số lượng vật phẩm trong kho, nếu số lượng bằng 0 thì vật phẩm sẽ bị xóa.

get\_item\_count(): Trả về số lượng của một loại vật phẩm cụ thể trong kho.

get\_item\_icon(): Trả về biểu tượng tương ứng với loại vật phẩm.

save(): Ghi nội dung của kho đồ xuống đĩa dưới dạng tệp inventory.tres để lưu trữ.

Ứng dụng:

Hệ thống này cho phép quản lý kho đồ trong game, theo dõi số lượng vật phẩm người chơi sở hữu và cung cấp cơ chế dễ dàng để thêm/bớt vật phẩm.

Các biểu tượng trực quan giúp hiển thị trực tiếp vật phẩm trong giao diện người dùng.

Kho đồ được lưu trữ dưới dạng tệp, cho phép phục hồi dữ liệu khi người chơi thoát và quay lại trò chơi.

A screenshot of a video game

Description automatically generated

Hình 3.36: Giao diện Inventory

Giao diện của inventory sẽ là một Hboxcontainer. Lớp này tên là UIInventory là một phần của giao diện người dùng (UI) trong trò chơi, có nhiệm vụ hiển thị các vật phẩm mà người chơi đã thu thập được. Trong lớp này, các vật phẩm được quản lý dưới dạng một danh sách đơn giản, được hiển thị dưới dạng các biểu tượng (icons). Mặc dù hệ thống hiện tại đơn giản, bạn có thể phát triển nó phức tạp hơn trong các bản cập nhật tương lai, tách biệt giữa dữ liệu (data) và giao diện (UI).

Các thành phần chính:

\_ITEM\_SCENE: Đây là một PackedScene, dùng để tải sẵn mẫu cho một mục trong danh sách inventory. Mẫu này có thể được sử dụng để tạo các mục mới mỗi khi người chơi thu thập một vật phẩm.

\_HIDDEN\_ITEMS: Đây là một danh sách các loại vật phẩm đặc biệt không được hiển thị trong giao diện inventory. Điều này có thể hữu ích nếu bạn muốn giấu những vật phẩm quan trọng hoặc vật phẩm bí mật không để người chơi thấy ngay.

\_ready(): Phương thức này được gọi khi đối tượng xuất hiện lần đầu trong cây cảnh (scene tree). Nó khởi tạo giao diện inventory bằng cách khôi phục trạng thái inventory từ dữ liệu đã lưu và cập nhật các vật phẩm được người chơi thu thập.

Kết nối tín hiệu item\_changed với hàm \_on\_inventory\_item\_changed() để cập nhật các thay đổi trong inventory khi người chơi nhận hoặc mất vật phẩm.

get\_ui\_item(item\_id: Inventory.ItemTypes): Phương thức này lặp qua các đối tượng con của UIInventory và kiểm tra xem có đối tượng nào trùng với item\_id đang được tìm kiếm không. Nếu có, nó sẽ trả về đối tượng đó.

\_update\_item(item\_id: Inventory.ItemTypes, inventory: Inventory): Đây là phương thức chính để cập nhật giao diện khi có thay đổi trong inventory:

+ Lấy số lượng: Sử dụng get\_item\_count() để lấy số lượng của vật phẩm có item\_id từ inventory.

+ Kiểm tra danh sách vật phẩm ẩn: Nếu item\_id nằm trong + \_HIDDEN\_ITEMS, phương thức sẽ kết thúc mà không thực hiện hành động nào.

+ Thêm hoặc cập nhật vật phẩm: Nếu số lượng lớn hơn 0 và vật phẩm chưa tồn tại trong giao diện, tạo ra một UIInventoryItem mới từ mẫu \_ITEM\_SCENE, thiết lập thuộc tính ID và biểu tượng cho nó, sau đó thêm vào giao diện.

+ Xóa vật phẩm: Nếu số lượng bằng 0 và vật phẩm đã tồn tại, gỡ bỏ vật phẩm khỏi giao diện bằng cách gọi queue\_free().

\_on\_inventory\_item\_changed(item\_type: Inventory.ItemTypes, inventory: Inventory): Phương thức này được gọi khi tín hiệu item\_changed được phát ra từ inventory. Nó gọi phương thức \_update\_item() để cập nhật vật phẩm trong giao diện.

UIInventory cung cấp một giao diện người dùng đơn giản để hiển thị các vật phẩm mà người chơi đã thu thập được trong trò chơi. Nó theo dõi trạng thái inventory và tự động cập nhật giao diện khi có thay đổi. Mặc dù hệ thống hiện tại chỉ hiển thị các vật phẩm một cách đơn giản, nó hoàn toàn có thể được mở rộng và phát triển thành một hệ thống phức tạp hơn trong tương lai.

### 3.20 Xây dựng level

Hệ thống Level Up là một cơ chế quan trọng trong trò chơi, cho phép nhân vật phát triển sức mạnh và khả năng của mình qua việc tích lũy kinh nghiệm (EXP). Dưới đây là các yếu tố cốt lõi của hệ thống Level Up được xây dựng trong mã của bạn:

- Tích lũy kinh nghiệm (EXP): Khi nhân vật thực hiện một hành động hoặc hoàn thành nhiệm vụ trong trò chơi, họ sẽ nhận được một lượng kinh nghiệm (EXP) nhất định.

Khi hàm add\_exp\_to\_player được gọi, lượng kinh nghiệm sẽ được cộng vào tổng EXP hiện tại của nhân vật. Nếu tổng số EXP đạt mức yêu cầu để lên cấp, hệ thống sẽ tự động gọi hàm level\_up để thực hiện quá trình lên cấp.

- Điều kiện lên cấp: Để lên cấp, nhân vật phải đạt được hoặc vượt qua exp\_to\_next\_level, tức là số lượng kinh nghiệm cần thiết để tiến lên cấp tiếp theo.

Nếu EXP đạt mức này, nhân vật sẽ lên cấp và exp\_to\_next\_level sẽ tăng lên theo công thức exp\_to\_next\_level \* 1.5, làm cho mỗi lần lên cấp sau đòi hỏi nhiều kinh nghiệm hơn, giúp quá trình thăng tiến ngày càng thử thách.

- Cải thiện chỉ số nhân vật: Mỗi lần lên cấp, các chỉ số của nhân vật sẽ được cải thiện, cụ thể như sau:

Knight: Khi lên cấp, sẽ được tăng 20 máu, 5 sức tấn công, và 5 phòng thủ. Điều này giúp Knight trở nên mạnh mẽ và bền bỉ hơn, phù hợp với vai trò của một chiến binh mạnh mẽ.

Mage: Khi lên cấp, sẽ tăng thêm 10 máu, 5 sức tấn công, và 1 năng lượng tối đa. Điều này làm cho Mage có khả năng sử dụng nhiều phép thuật hơn, phù hợp với vai trò của một pháp sư tập trung vào sức mạnh phép thuật.

Các chỉ số này có thể được tùy chỉnh thêm để phù hợp hơn với yêu cầu cụ thể của trò chơi.

- Khôi phục máu và năng lượng: Sau khi lên cấp, nhân vật sẽ được hồi phục toàn bộ máu và năng lượng về mức tối đa. Điều này không chỉ giúp nhân vật trở nên mạnh mẽ hơn mà còn tạo động lực cho người chơi tiếp tục thử thách các nhiệm vụ cao hơn mà không lo ngại về tình trạng kiệt sức của nhân vật.

### 3.21 Xây dựng chức năng Buff và Debuff

- **Mục tiêu:** Chức năng Buff và Debuff được thêm vào để nâng cao trải nghiệm chiến đấu trong game, giúp người chơi và AI có thể sử dụng chiến lược linh hoạt hơn. Buff giúp tăng cường các chỉ số tạm thời (như Attack, Defense, Speed), trong khi Debuff làm suy giảm chỉ số đối phương trong một khoảng thời gian ngắn.

- **Ý tưởng triển khai:**

+ Buff: Tăng các chỉ số như Attack, Defense, hoặc Speed của nhân vật trong một thời gian giới hạn.

+ Debuff: Giảm chỉ số tương tự của đối phương, làm yếu sức mạnh hoặc khả năng phòng thủ của họ.

Thời gian tồn tại của Buff/Debuff sẽ giảm dần và tự động kết thúc sau khi hết hiệu lực.

**- Cách thực hiện:**

1. Cập nhật hệ thống chỉ số (Stats):

+ Tạo một danh sách chứa các hiệu ứng Buff/Debuff hiện tại.

+ Mỗi hiệu ứng sẽ bao gồm:

* Chỉ số bị ảnh hưởng (Attack, Defense, Speed, v.v.).
* Giá trị thay đổi (dương cho Buff, âm cho Debuff).
* Thời gian tồn tại (theo lượt).

+ Hệ thống sẽ tự động áp dụng hoặc hủy bỏ Buff/Debuff khi hết thời gian.

2. Quản lý Buff/Debuff trong nhân vật:

+ Mỗi nhân vật sẽ có một cơ chế để áp dụng Buff/Debuff khi sử dụng kỹ năng hoặc chịu ảnh hưởng từ đối thủ.

+ Ví dụ:

Buff: Tăng 20 Attack trong 10 giây.

Debuff: Giảm 15 Defense trong 5 giây.

3. Cài đặt kỹ năng sử dụng Buff/Debuff:

+ Một số kỹ năng sẽ có tác dụng Buff cho đồng minh hoặc Debuff cho đối thủ.

+ Ví dụ:

Kỹ năng "Strengthen" tăng Attack cho đồng đội.

Kỹ năng "Weaken" làm giảm Defense của đối thủ.

4. Hiển thị trạng thái Buff/Debuff:

Sử dụng giao diện Hboxcontainer để hiển thị biểu tượng của Buff/Debuff trên nhân vật.

Ví dụ:

Biểu tượng màu xanh để biểu thị Buff.

Biểu tượng màu đỏ để biểu thị Debuff.

Triển khai:

A cartoon of a cat

Description automatically generated

Hình 3.37: Buff và debuff sẽ được hiển thị bằng icon kèm số lượt hiệu lực nằm phí trên Battler.

Buff: Khi người chơi sử dụng kỹ năng "Strengthen," đồng minh được tăng thêm 15 điểm Attack trong 10 giây.

Debuff: Khi kẻ địch trúng kỹ năng "Weaken," Defense của chúng bị giảm 20 điểm trong 5 giây.

# CHƯƠNG 4: ĐÁNH GIÁ KẾT QUẢ

- Kết quả đạt được:

+ Hệ thống chiến đấu: Dự án đã hoàn thiện hệ thống chiến đấu theo lượt với các yếu tố đa dạng, từ việc lựa chọn kỹ năng, mục tiêu đến tính toán sát thương, phòng thủ và tốc độ. Cơ chế này giúp game trở nên hấp dẫn và đòi hỏi chiến thuật từ người chơi, giúp họ có những trải nghiệm phong phú.

+ Tối ưu hóa trải nghiệm người chơi: Các giao diện như quản lý trang bị và kho đồ đều có thể điều chỉnh linh hoạt, tạo ra sự tiện lợi cho người chơi. Người chơi có thể dễ dàng thay đổi trang bị và kiểm soát chỉ số nhân vật, từ đó đưa ra chiến thuật phù hợp.

+ Xử lý tình huống trong game: Tính năng như kết thúc trận chiến, sự kiện thắng/thua và cập nhật chỉ số sau mỗi trận đều hoạt động chính xác. Hệ thống đảm bảo việc kiểm tra tình trạng nhân vật và cho phép quản lý lượt đánh một cách hiệu quả.

+ Tích hợp hệ thống lên cấp và nâng cấp nhân vật: Hệ thống lên cấp giúp nhân vật của người chơi phát triển qua các trận đấu, tăng cường chỉ số và mở khóa kỹ năng mới. Điều này tạo động lực và thách thức cho người chơi khi họ có cơ hội xây dựng chiến thuật cá nhân thông qua việc nâng cấp chỉ số phù hợp.

+ Khả năng mở rộng hệ thống kỹ năng và trang bị: Dự án đã thiết kế hệ thống kỹ năng và trang bị với khả năng mở rộng dễ dàng. Người phát triển có thể dễ dàng bổ sung thêm các kỹ năng hoặc vật phẩm mới mà không cần thay đổi nhiều trong cấu trúc code, giúp dự án có tiềm năng mở rộng trong tương lai và đáp ứng nhu cầu của người chơi.

+ Tối ưu hóa hiệu suất: Dự án đã đạt được sự ổn định và mượt mà trong quá trình chạy, đặc biệt là khi xử lý các trận đấu và cập nhật trạng thái của nhân vật. Các tối ưu hóa về mặt hiệu suất giúp trò chơi vận hành ổn định ngay cả trên các thiết bị có cấu hình trung bình.

+ Trải nghiệm người chơi: Dự án đã chú trọng vào việc tạo ra trải nghiệm người chơi hấp dẫn thông qua các giao diện trực quan và đơn giản. Các bảng chọn kỹ năng, bảng chỉ số nhân vật và giao diện trang bị đều được thiết kế dễ hiểu, giúp người chơi dễ dàng thao tác mà không cần hướng dẫn quá nhiều.

+ Quản lý kho đồ (Inventory): Tính năng quản lý kho đồ hoạt động tốt, cho phép người chơi dễ dàng sử dụng và trang bị vật phẩm. Giao diện Inventory đã hỗ trợ đầy đủ các chức năng cơ bản như thêm, bỏ và sử dụng vật phẩm. Việc mở rộng thêm các tùy chọn hiển thị cho phép người chơi xem các chi tiết của vật phẩm sẽ giúp cải thiện tính trực quan.

+ Hệ thống nhiệm vụ: Hệ thống nhiệm vụ hoạt động mượt mà, cho phép người chơi nhận và hoàn thành nhiệm vụ theo cốt truyện. Tính năng theo dõi và cập nhật tiến trình của nhiệm vụ cũng được triển khai tốt, giúp người chơi dễ dàng theo dõi nhiệm vụ hiện tại và phần thưởng.

+ Tốc độ tải và phản hồi: Quá trình tải game và chuyển cảnh diễn ra nhanh chóng, không có hiện tượng giật lag đáng kể. Combat được tối ưu hóa tốt với hiệu ứng mượt mà ngay cả khi có nhiều đối tượng xuất hiện trên màn hình. Tuy nhiên, để đảm bảo ổn định khi tải các tài nguyên lớn hơn, có thể xem xét việc tối ưu bộ nhớ bằng cách nén ảnh hoặc sử dụng các kỹ thuật tải chậm (lazy loading).

+ Sử dụng tài nguyên: Mức sử dụng CPU và RAM ổn định, tuy nhiên cần tiếp tục giám sát trong các tình huống chiến đấu lớn. Có thể cải thiện thêm bằng cách tối ưu hóa các hiệu ứng đồ họa khi có nhiều kỹ năng được thực hiện liên tiếp.

- Hạn chế và hướng phát triển:

+ Mặc dù dự án đã hoàn thành đúng tiến độ và đạt được các mục tiêu ban đầu, một số hạn chế vẫn tồn tại, như khả năng mở rộng thêm các loại kỹ năng, hay cải tiến giao diện để tạo ra trải nghiệm mượt mà hơn. Trong tương lai, dự án có thể tiếp tục phát triển các tính năng mới như hệ thống nhiệm vụ, cốt truyện mở rộng, và thêm các chế độ chơi khác để tăng thêm độ phong phú cho trò chơi.

+ Tựa game theo hướng turn base thực tế cần nhiều animation hơn. Điều này ảnh hưởng trực tiếp đến trải nghiệm người chơi khi các hành động trong game như tấn công, phòng thủ hoặc sử dụng kỹ năng chỉ thể hiện qua các thay đổi trạng thái đơn giản thay vì có hiệu ứng động hấp dẫn. Sự thiếu hụt này có thể làm giảm sức hấp dẫn của trò chơi và khiến các pha chiến đấu trở nên ít sinh động.

Kết luận: Dự án đã mang lại những kết quả khả quan, thể hiện rõ khả năng áp dụng lý thuyết vào thực tế trong lĩnh vực phát triển game RPG. Những kỹ năng và kinh nghiệm thu được sẽ là nền tảng vững chắc cho những dự án tiếp theo.

KẾT LUẬN

Dự án xây dựng game chiến thuật theo lượt (Turn-Based RPG) đã đạt được mục tiêu ban đầu đề ra, thể hiện khả năng kết hợp giữa công nghệ và sự sáng tạo. Qua việc sử dụng Godot Engine 4, dự án không chỉ tận dụng được các tính năng mạnh mẽ của công cụ mã nguồn mở mà còn khắc phục được những thách thức trong quá trình phát triển như tối ưu hóa giao diện, xây dựng hệ thống chiến đấu, và hoàn thiện cốt truyện.

Kết quả của dự án không chỉ mang lại một trò chơi hoàn chỉnh với các chức năng nổi bật như hệ thống chiến đấu, quản lý vật phẩm, và tương tác NPC, mà còn mở ra tiềm năng phát triển thêm nhiều nội dung mới. Những bài học và kinh nghiệm thu được từ dự án sẽ là nền tảng quý giá để tiếp tục thực hiện các dự án trong tương lai.

Dự án khẳng định vai trò của công nghệ và sự đổi mới trong việc tạo ra các sản phẩm giải trí hấp dẫn, đồng thời thể hiện cam kết với việc không ngừng học hỏi và phát triển bản thân. Hy vọng trò chơi này sẽ mang lại những phút giây thư giãn và giá trị giải trí đến cộng đồng người chơi.

TÀI LIỆU THAM KHẢO

*[1] Dialogic-Godot, "Dialogic - Create Dialogs, Visual Novels, RPGs, and manage Characters with Godot," GitHub Repository, 2024. [Trực tuyến]. Có sẵn: https://github.com/dialogic-godot/dialogic.*

*[2] YouTube, "Video Tutorial," YouTube, 2024. [Trực tuyến]. Có sẵn: https://www.youtube.com/watch?v=X3J0fSodKgs.*

*[3] YouTube, "Video Tutorial," YouTube, 2024. [Trực tuyến]. Có sẵn: https://www.youtube.com/watch?v=FV4JkwI4OF4.*

*[4] Godot Engine, "Godot Engine Documentation," [Trực tuyến]. Available:* [*https://docs.godotengine.org/en/stable/index.html*](https://docs.godotengine.org/en/stable/index.html)*.*

**Kết quả turnitin**

*(In trang đầu tiên trong báo cáo check turnitin và đóng vào cuối quyển)*

A screenshot of a computer

Description automatically generated