TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CNTT & TT

——————— * ———————

BÀI TẬP LỚN

MÔN: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Đề tài: **Xây dựng game nhập vai 2D RPG**

Nhóm: **11**
Sinh viên thực hiện: **Nguyễn Trọng Tuệ**
MSSV: **20194710**
Lớp: **130930**
Giảng viên hướng dẫn: **ThS. Nguyễn Mạnh Tuấn**

Hà Nội, tháng 7 năm 2022

MỤC LỤC

LỜI NÓI ĐẦU

Với sự phát triển mạnh mẽ và không ngừng của công nghệ trong kỷ nguyên số hiện nay, trò chơi điện tử luôn là một trong những mảng miếng được rất nhiều các tập đoàn công nghệ lớn trên thế giới quan tâm và đầu tư mạnh mẽ. Từ những ngày đầu khi những trò chơi với đồ họa bitmap thô sơ trên các máy tính MS.DOS, dần dần tiến đến sự phát triển của các hệ máy chơi game nổi tiếng suốt bao thập kỷ như Nintendo, PS, Xbox, … rồi sự phát triển không ngừng của thiết bị phần cứng cũng như mạng Internet, thiết bị di động … khiến cho các trò chơi ngày càng hấp dẫn về mọi mặt: chế độ trực tuyến nhiều người chơi, đồ họa 3D với kỹ xảo bắt mắt, hay thậm chí là các trò chơi VR, AR, điển hình như PokemonGO đem lại những trải nghiệm vô tiền khoáng hậu cho người chơi trò chơi điện tử. Có thể nói trò chơi điện tử đã và luôn là một món ăn tinh thần khó có thể thiếu được trong cuộc sống của chúng ta.

Bên cạnh các trò chơi với nền đồ họa khủng, hệ thống nhân vật, vật phẩm đồ sộ, lượng người chơi đông đảo, … Những trò chơi mang thiên hướng retro, hoài niệm về những ngày xưa cũ, thiết kế trên ý tưởng nền đồ họa bitmap, gameplay đơn giản vẫn luôn được một bộ phận người chơi không nhỏ yêu thích với rất nhiều sản phẩm tốt trên thị trường game cạnh tranh khốc liệt.

Với niềm yêu thích các tựa game hoài cổ trên nền đồ họa bitmap như thế, cũng như mong ước muốn có thể làm ra sản phẩm game của bản thân từ những ngày thơ ấu, em đã quyết định lựa chọn đề tài số 1 "Xây dựng game nhập vai 2D đơn giản" là đề tài sẽ đi cùng em trong môn học Lập trình hướng đối tượng này.

Sẽ khó tránh khỏi những thiếu sót, nhầm lẫn trong quá trình em thực hiện project này, cũng như việc hoàn thành project này với nhóm duy nhất một thành viên cũng khá vất vả vì khối lượng công việc là tương đối lớn. Tuy vậy, em đã cố gắng hết sức để có thể hoàn thành project này một cách tốt nhất trong khả năng của bản thân mình thông qua quá trình tích lũy, tìm tòi, học hỏi các kiến thức của môn học Lập trình hướng đối tượng. Một lần nữa em xin chân thành cảm ơn thầy đã đem đến những kiến thức hữu ích và hấp dẫn về môn học này đến cho em.

PHÂN CÔNG THÀNH VIÊN TRONG NHÓM

Họ và tên	**Email**	**Điện thoại**	**Công việc thực hiện**	**Đánh giá**
Nguyễn Trọng Tuệ	tue.nt194710@sis.hust.edu.vn	0946103302	- Xây dựng, phân tích, thiết kế bài toán (2D RPG) - Lập trình, xử lý đồ họa, thiết kế gameplay, giao diện của trò chơi - Kiểm thử, gỡ lỗi trò chơi - Viết báo cáo về sản phẩm trò chơi đã xây dựng được	

CHƯƠNG 1. KHẢO SÁT, ĐẶC TẢ YÊU CẦU BÀI TOÁN

1.1. Mô tả yêu cầu bài toán

Bài toán tổng quát được đặt ra với chủ đề này là làm thế nào để có thể xây dựng một trò chơi nhập vai đơn giản đáp ứng đủ những nhu cầu về mặt giải trí của người chơi, cũng như có thể vận hành một cách trơn tru, ít lỗi.

Để giải quyết bài toán này, em đã có tìm hiểu qua cơ chế gameplay một số game 2D thể loại nhập vai, kết hợp với những trải nghiệm chơi game đã có của bản thân. Qua đó, mô tả yêu cầu của bài toán cụ thể như sau:

- Trò chơi sẽ có người chơi, quái vật chiến đấu với nhau. Đặc biệt, trong trò chơi sẽ xuất hiện boss (trùm cuối). Nhiệm vụ của người chơi là phải nâng cấp sức mạnh để có thể chiến thắng boss, qua đó chiến thắng trò chơi. Ngược lại, nếu người chơi bị quái vật giết chết trước khi chạm trán boss, hoặc là bị boss hạ gục thì người chơi sẽ thua cuộc (game over).

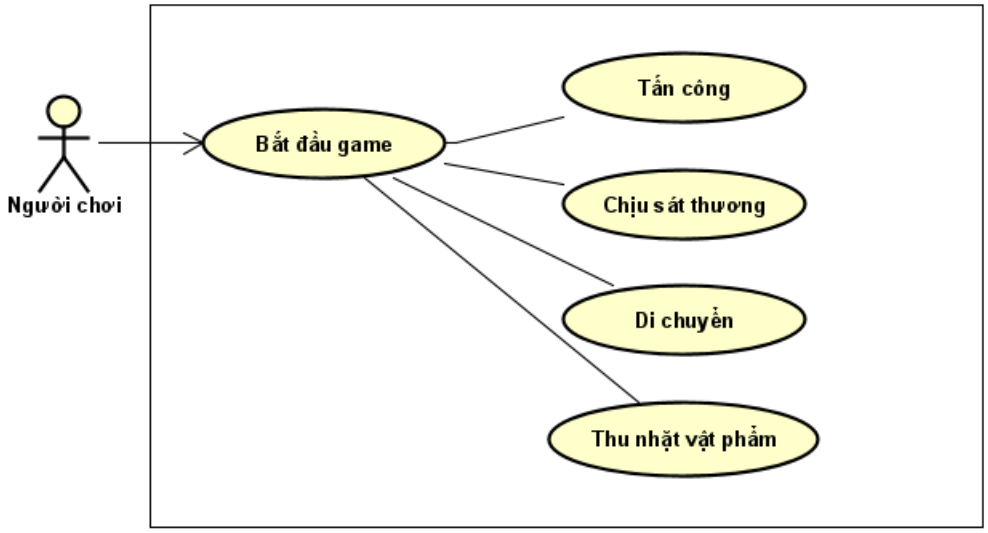
- Trò chơi sẽ tích hợp hệ thống cấp độ(level), máu, tấn công, phòng thủ, tốc độ di chuyển tương ứng cho cả đối tượng người chơi và quái vật để cả 2 phía có thể thực hiện tương tác lẫn nhau trong trò chơi (tấn công, hạ gục). Đặc biệt, trò chơi xây dựng một số kĩ năng tấn công đặc biệt cho cả đối tượng người chơi và quái vật, cùng với đó là hệ thống vật phẩm hữu ích trong trò chơi, giúp trò chơi trở nên thú vị, kịch tính và hấp dẫn hơn.

- Trò chơi sẽ có 2 mức độ là dễ và khó. Tùy thuộc vào khả năng của bản thân, người chơi có thể lựa chọn độ khó phù hợp. Lựa chọn mức độ khó đồng nghĩa với việc quái vật sẽ có nhiều máu hơn, chỉ số tấn công và phòng thủ cao hơn, tốc độ di chuyển cao hơn, qua đó sẽ gây nhiều khó khăn hơn cho người chơi

- Trò chơi sẽ có một vài map khác nhau, cũng như hệ thống sinh quái tự động ngẫu nhiên, giúp trò chơi trở nên khó đoán hơn.

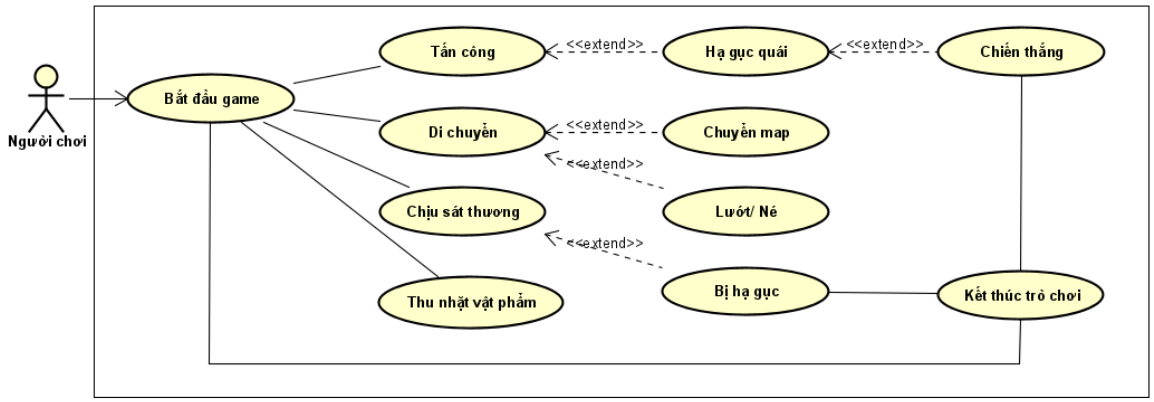
1.2. Biểu đồ use case

1.2.1. Biểu đồ use case tổng quan



- Thông qua việc mô tả yêu cầu bài toán, biểu đồ use case tổng quan được xây dựng nhằm mục đích khái quát hóa những chức năng tổng quát của nhóm đối tượng người chơi trong trò chơi như đã đề cập ở mục trước như tấn công, chịu sát thương, di chuyển, thu nhặt vật phẩm (đối với người chơi).

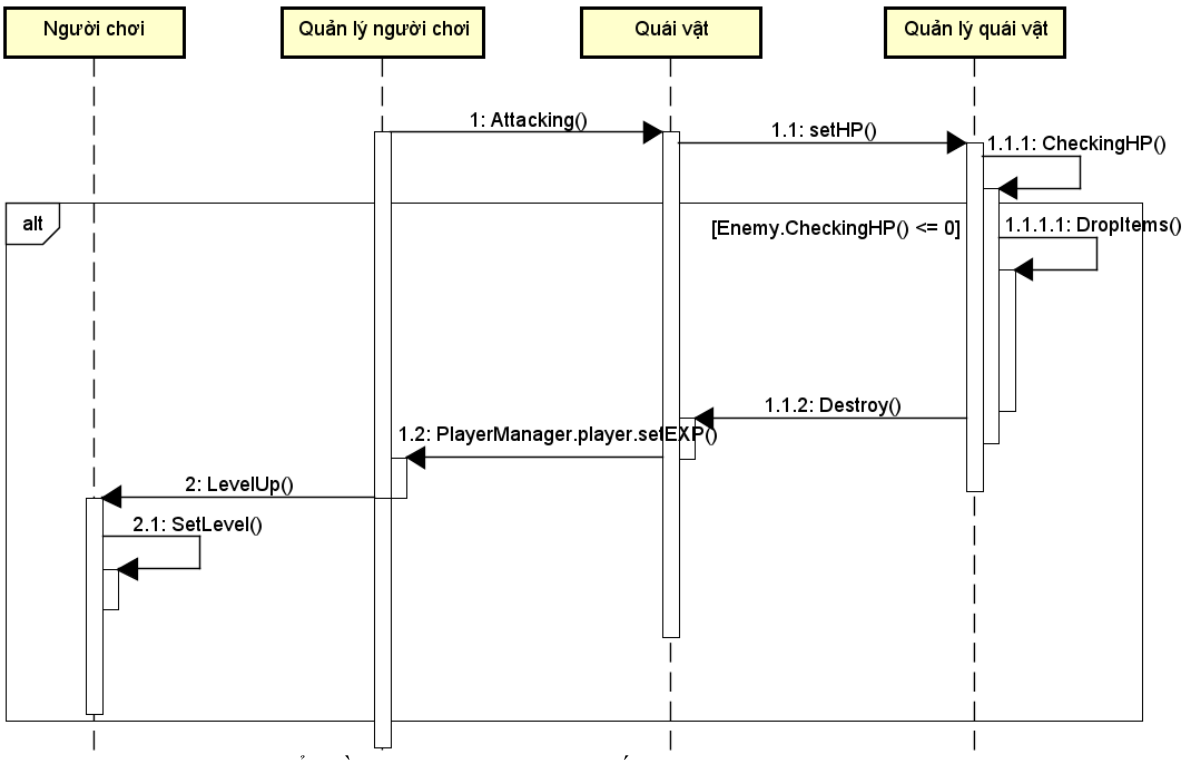
1.2.2. Biểu đồ use case phân rã mức 2



- Về phía người chơi, khi bắt đầu trò người chơi có thể di chuyển, tấn công kẻ địch, cũng như đồng thời có thể thu nhặt vật phẩm để có thể thăng tiến sức mạnh thông qua cập nhật chỉ số. Tương tự như thế đối với các đối tượng quái vật, tuy vậy có một sự khác biệt nỏ, đó là việc quái vật sẽ giữ nguyên chỉ số của mình, không có sự thay đổi trong suốt toàn bộ quá trình game.
- Trò chơi kết thúc khi người chơi bị hạ gục, hoặc boss bị hạ gục, dẫn đến kết thúc trò chơi. Người chơi có thể chọn bắt đầu lại trò chơi hoặc thóat trò chơi.

CHƯƠNG 2. PHÂN TÍCH VÀ THIẾT KẾ BÀI TOÁN

2.1 Biểu đồ trình tự



Hình 2.2.1: Biểu đồ trình tự của chức năng tấn công được thực hiện bởi người chơi

Biểu đồ bên trên biểu diễn khái quát trình tự thực hiện đòn tấn công của người chơi:

- Thông qua lớp Quản lý người chơi (PlayerManager), người chơi thực hiện tấn công quái vật. Lúc này, property (EnemyStatistic) enemy.HP được cập nhật giá trị mới.
- Phương thức CheckingHP() ở lớp quản lý quái vật được gọi để kiểm tra xem ngưỡng máu của quái vật đã xuống dưới 0 chưa. Nếu thỏa mãn thì quái vật sẽ có tỷ lệ ngẫu nhiên sinh ra vật phẩm tại vị trí hiện tại của nó, sau đó bị phá hủy bởi Destroy().
- Thực hiện cập nhật kinh nghiệm (Exp) của người chơi, sau đó gọi đến LevelUp() để kiểm tra xem người chơi đã đủ kinh nghiệm lên cấp tiếp theo chưa để có thể SetLevel()?

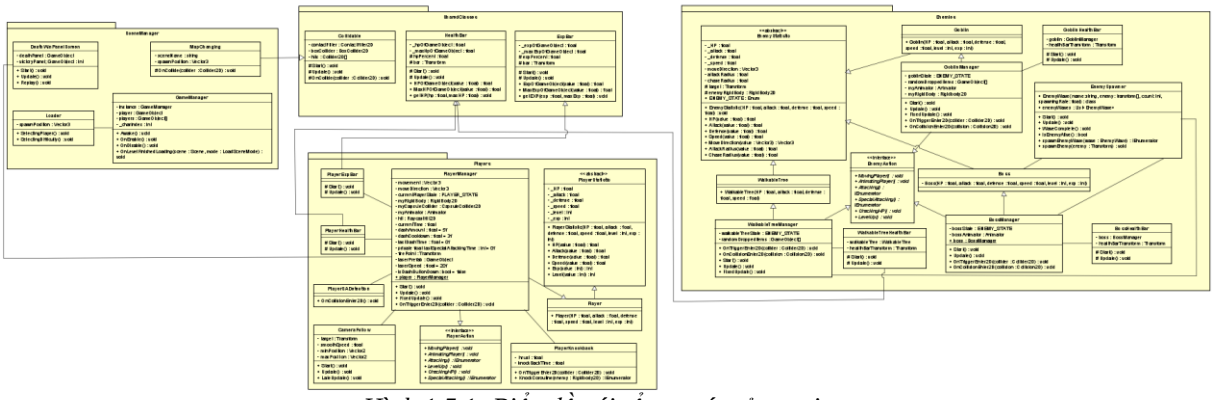
2.2 Biểu đồ gói và biểu đồ lớp

Để có thể tương tác tốt với nền tảng Unity (sử dụng ngôn ngữ C#) mà em sẽ đi vào giải thích chi tiết hơn ở phần sau, tổ chức của các gói, các lớp trong project sẽ được em triển khai như sau:

- Các lớp trong hầu hết các lớp trong project sẽ được đặt chỉ định truy cập là public và không đặt trong bất kỳ namespace nào. Để tránh sự rắc rối cũng như

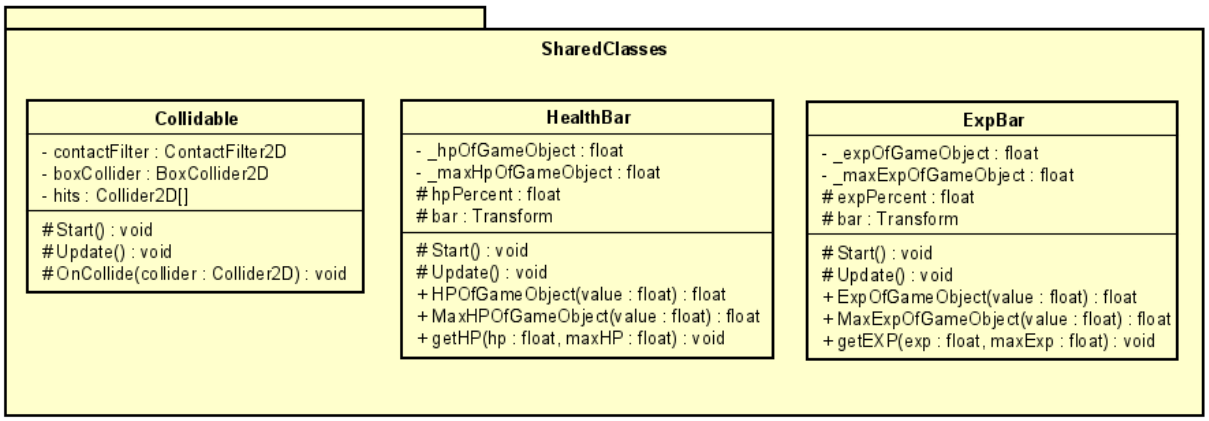
nhầm lẫn xảy ra, các lớp cùng hướng tới một nhóm đối tượng sẽ được đặt trong cùng thư mục.

- Đồng thời, tất cả mọi lớp trong project sẽ đều kế thừa từ lớp UnityEngine.MonoBehaviour của namespace UnityEngine, không trực tiếp thì sẽ thông qua lớp cha để kế thừa lớp MonoBehaviour này vì đây là lớp mà mọi file mã nguồn C# trong project sử dụng Unity đều phải kế thừa nó. Nên ở đây em sẽ mặc định là mọi lớp đã kế thừa class Unity.MonoBehaviour và không đưa cụ thể class này vào trong biểu đồ lớp/ gói.
- Bên cạnh đó, mọi lớp sẽ đều sử dụng namespace System.Collections, thuận tiện cho việc xử lý tính toán liên quan đến đối tượng thông qua các Collections có sẵn của C#.
- Hầu hết tất cả các lớp/ gói trong project đều được gắn cùng đối tượng sẽ thực hiện chức năng tương ứng trong Unity. Do vậy, gần như mọi lớp trong project sẽ chỉ chủ yếu thực hiện việc xử lý các tính toán số học, vật lý hay kiểm tra tính đúng đắn của hoạt ảnh động của một đối tượng nào đó. Phần xử lý đồ họa nền chủ yếu sẽ do Unity đảm nhận.
- Sau đây, em sẽ giới thiệu tất cả các lớp trong project phân chia theo thư mục (nhóm các lớp hướng tới cùng một nhóm đối tượng hoặc cùng có mục đích thực hiện một nhóm nghiệp vụ nào đó) thông qua biểu đồ gói và biểu đồ lớp của project:



Hình 1.5.1: Biểu đồ gói tổng quát của project

a) <u>SharedClasses</u>

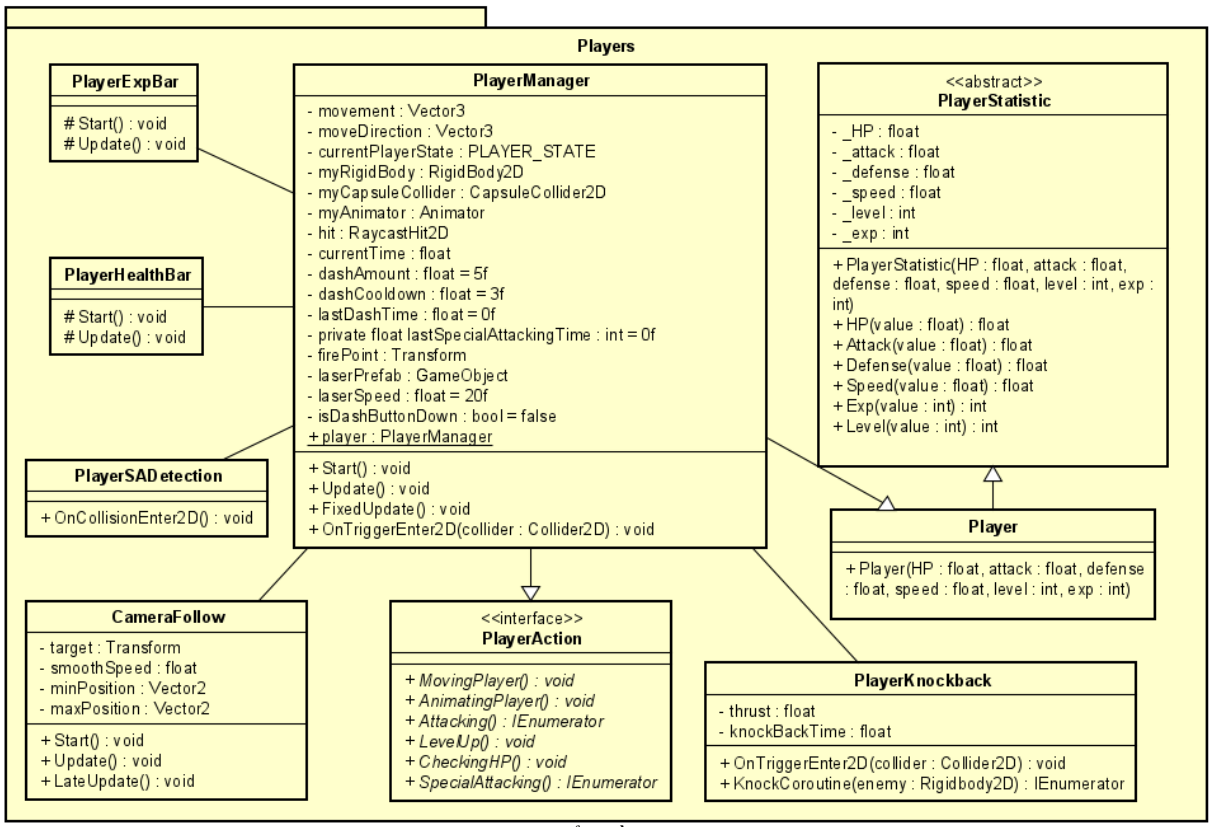


Hình 1.5.2: Biểu đồ lớp của thư mục SharedClasses

Thư mục SharedClasses chứa 3 lớp là Collidable, HealthBar, ExpBar. Trong đó:

- Collidable: xử lý các va chạm có thể dẫn đến hành vi hoặc sự kiện cụ thể nào đó (ví dụ như chuyển map)
- HealthBar: Lớp hiển thị thanh máu của đối tượng mà lớp của đối tượng đó kế thừa từ lớp này, cập nhật liên tục mỗi frame thông qua phương thức MonoBehaviour.Update()
- ExpBar: Lớp hiển thị thanh kinh nghiệm của đối tượng mà lớp của đối tượng đó kế thừa từ lớp này, cập nhật liên tục mỗi frame thông qua phương thức MonoBehaviour.Update()

b) Players



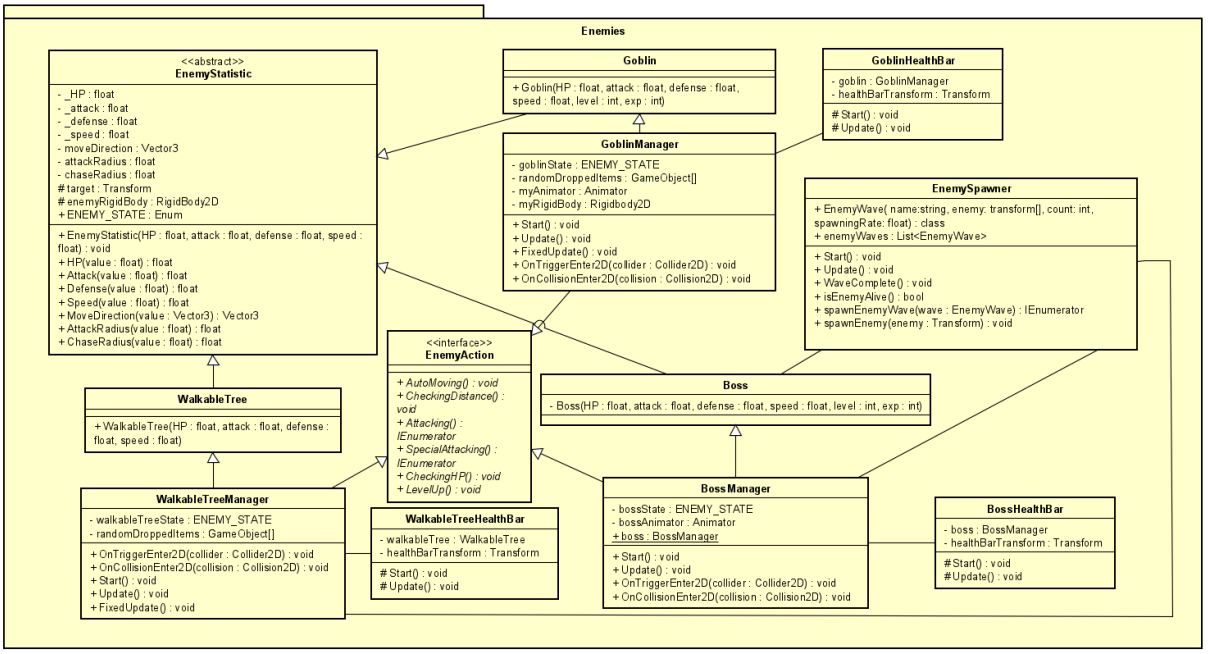
Hình 1.5.3: Biểu đồ lớp của Players

Thư mục Players chứa các lớp/ giao diện cùng chung mục đích xử lý các thao tác, hành vi của người chơi đối với đối tượng nhân vật trong trò chơi. Trong đó nhiệm vụ của các lớp/ giao diện như sau:

- PlayerStatistic: lớp trừu tượng, chứa các thuộc tính/ thông số cơ bản của người chơi chưa được cài đặt cụ thể.
- Player: lớp kế thừa từ PlayerStatistic, thực hiện trực tiếp việc thiết lập thông số của người chơi.
- PlayerAction: Giao diện cung cấp nhóm các phương thức/ hành vi mà đối tượng người chơi sẽ phải cài đặt.
- PlayerKnockback: lớp quản lý cơ chế đánh trả(knockback) thông qua tương tác vật lý của người chơi lên quái vật.
- PlayerHealthBar: lớp quản lý thanh máu của người chơi, kế thừa từ HealthBar.
- PlayerExpbar: lớp quản lý thanh kinh nghiệm của người chơi, kế thừa từ ExpBar.

- PlayerSADetection: lớp quản lý các sự kiện liên quan đến đòn tấn công đặc biệt của người chơi.
- CameraFollow: lớp quản lý camera của chương trình (hướng camera của game vào đối tượng người chơi hiện hữu trong game).
- PlayerManager: Kế thừa từ Player và PlayerAction, là lớp thực hiện quản lý cũng như thực hiện thực thi hầu hết tất cả các hành vi, trạng thái của người chơi cũng như xử lý các chuỗi sự kiện liên quan. Đây chính là lớp quan trọng nhất trong việc xử lý đối tượng người chơi.

c) Enemies

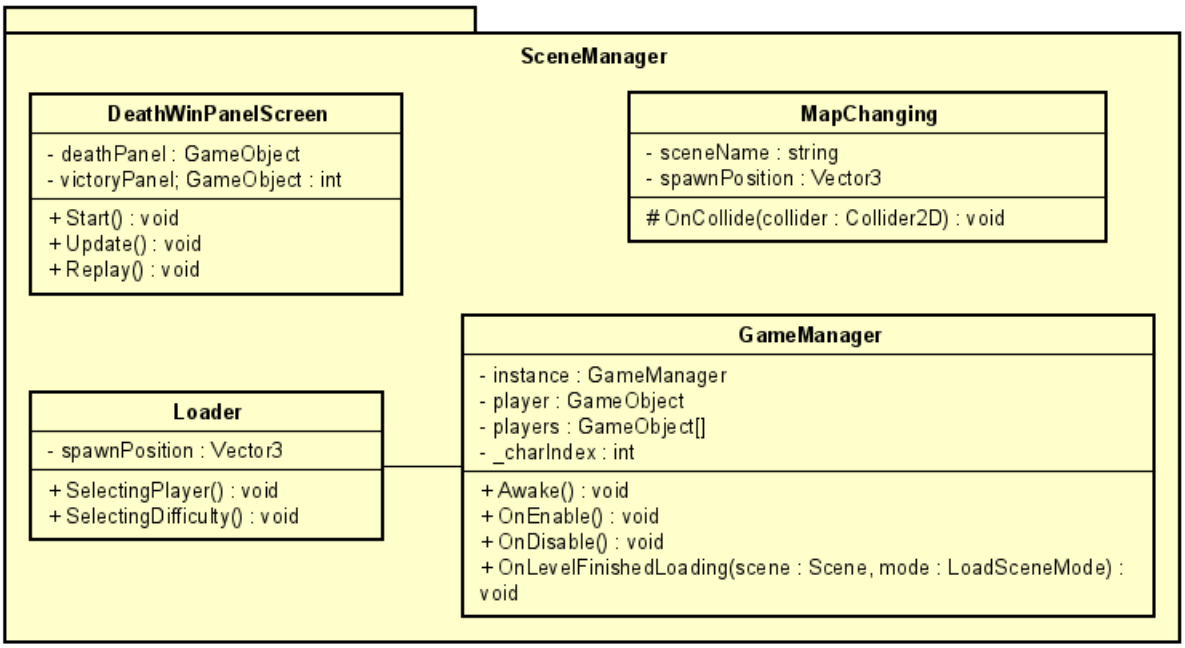


Hình 1.5.4: Biểu đồ lớp của Enemies

Thư mục Enemies chứa các lớp xử lý các tính toán toán học, vật lý cũng như quản lý các hành vi, sự kiện liên quan đến các đối tượng quái vật, cụ thể như sau:

- EnemyStatistic: lớp trừu tượng, chứa các thuộc tính/ thông số cơ bản của quái vật chưa được cài đặt cụ thể.
- EnemyAction: Giao diện cung cấp nhóm các phương thức/ hành vi mà đối tượng người chơi sẽ phải cài đặt. Tùy theo mỗi đối tượng quái vật thừa kế giao diện này mà có cách xử lý các phương thức này theo những cách khác nhau, thể hiện tính đa hình của lập trình hướng đối tượng một các rõ ràng.
- WalkableTree, Goblin, Boss: các lớp kế thừa từ EnemyStatistic, thực hiện trực tiếp việc thiết lập thông số của người chơi.
- WalkableTreeHealthBar, GoblinHealthBar, BossHealthBar: lớp quản lý thanh máu của đối tượng quái vật, kế thừa từ HealthBar trong SharedClasses.
- WalkableTreeManager, GoblinHealthManager, BossHealthManager: Kế thừa từ lớp cha chứa thông số thuộc tính của đối tượng quái vật đó và PlayerAction, là lớp thực hiện quản lý cũng như thực hiện thực thi hầu hết tất cả các hành vi, trạng thái của quái vật cũng như xử lý các chuỗi sự kiện liên quan. Đây chính là lớp quan trọng nhất trong việc xử lý đối tượng quái vật.

d) SceneManager



Hình 1.5.5: Biểu đồ lớp của SceneManager

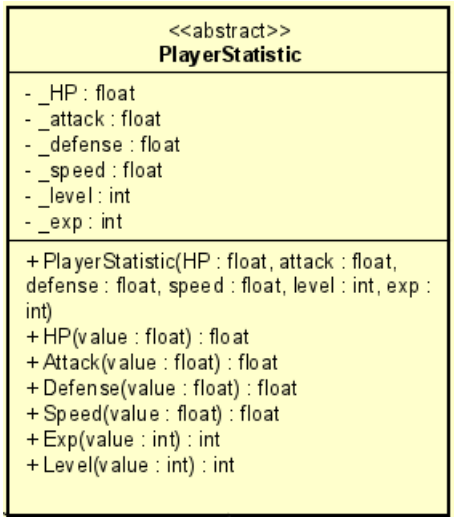
Thư mục SceneManager chứa các lớp cùng chung nhóm nhiệm vụ xử lý các thao tác chuyển cảnh, hiển thị thông báo, cũng như xử lý một số sự kiện nhỏ liên quan đến việc khởi tạo hoặc hủy người chơi. Cụ thể:

- Loader: xử lý sự kiện load giao diện khởi đầu trò chơi.
- GameManager: xử lý việc chọn đối tượng nhân vật nào để khởi tạo, cũng như thực hiện kỹ thuật Singleton Pattern để xử lý tốt việc chuyển dịch người chơi giữa các bản đồ khác nhau.
- MapChanging: Kế thừa từ Collidable, xử lý sự kiện chuyển bản đồ của người chơi.
- DeathWinPanelScreen: xử lý sự kiện hiển thị màn hình kết thúc trò chơi (khi người chơi chiến thắng hay thất bại).

⇨ Bên cạnh đó, nền tảng Unity cũng hỗ trợ việc thiết kế giao diện, xử lý đồ họa, … thông qua hình thức hướng đối tượng kiểu GameObject, em sẽ đi vào chi tiết hơn vấn đề này ở phần công nghệ sử dụng để thiết kế trò chơi.

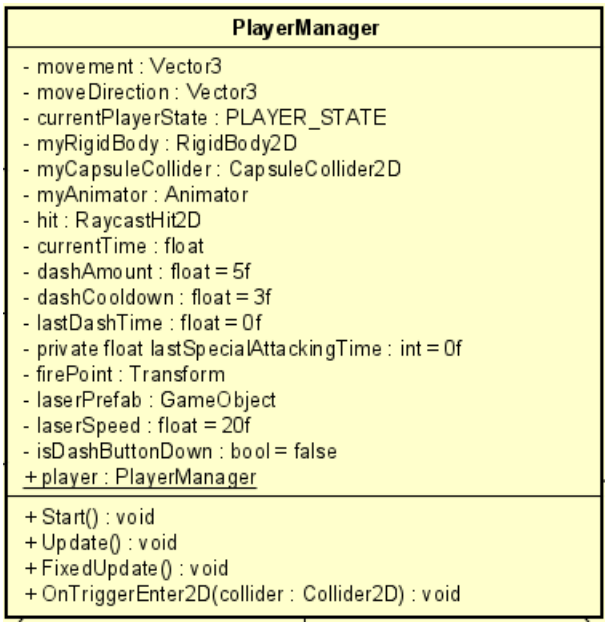
2.3 Thiết kế chi tiết lớp

Ở phần này, em sẽ đi vào thiết kế chi tiếp của một số lớp hay interface mang tính quyết định đến sự vận hành của trò chơi.



Hình 2.3.1: Lớp PlayerStatistic

- Đây là lớp trừu tượng(abstract class) đưa ra những thuộc tính tiên quyết mà mỗi đối tượng kiểu nhân vật sẽ phải có như máu(_HP), tấn công(_attack), phòng thủ(_defense), tốc độ di chuyern(_speed), cấp độ(_level), lượng kinh nghiệm(_exp). Vì là lớp trừu tượng nên sẽ không thể khởi tạo bất cứ đối tượng nào thể hiện cho lớp này, muốn sở hữu các Properties thiết lập các thuộc tính này thì người lập trình sẽ phải kế thừa lớp này để khởi tạo đối tượng có kế thừa từ PlayerStatistic. Các Properties ở đây đại diện cho tính trừu tượng của Lập trình hướng đối tượng, che giấu đi những cái cốt lõi phức tạp, chỉ đưa ra phương thức để người dùng sử dụng chúng.



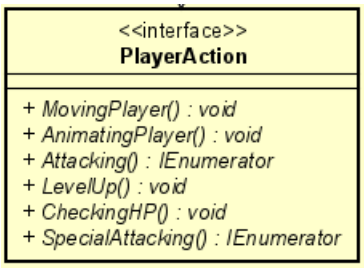
Hình 2.3.2: Lớp PlayerManager

- Lớp PlayerManager là một trong những lớp quan trọng nhất của trò chơi vì đây sẽ là lớp chính chịu trách nhiệm quản lý cũng như xử lý các hành vi, sự kiện của người chơi. Đây cũng là lớp chịu trách nhiệm chính cho việc tương tác với đối tượng người chơi mang kiểu GameObject trên nền tảng Unity.

- Hầu hết tất cả các thuộc tính của lớp sẽ được thiết lập chỉ định truy cập là private, nhằm tránh sự truy cập trái phép không mong muốn từ bên ngoài. Việc xử lý các thuộc tính này sẽ được thực hiện qua các phương thức public của lớp, điều này thể hiện rất rõ tính đóng gói trong Lập trình hướng đối tượng. Mô tả chi tiết các thuộc tính của lớp PlayerManager:
    - movement: thực hiện nhận giá trị kiểu Vector3 từ Input.GetKey, qua đó mô phỏng hướng di chuyển hiện tại của người chơi.
    - moveDirection: đưa giá trị của Vector3 được tiêu chuẩn hóa về giá trị 1, qua đó sẽ loại bỏ sự chênh lệch cự ly di chuyển giữa các hướng của người chơi. Giá trị đó được lưu trong thuộc tính này.
    - currentPlayerState: lưu lại trạng thái hiện tại của người chơi, ví dụ như đang đứng im(IDLE), di chuyển(WALK), tấn công(ATTACK), lướt/né (DASH), …
    - myRigidBody: thuộc tính sẽ nhận component(thành phần) cơ thể vật lý Rigidbody2D, đây là một trong những thuộc tính quan trọng nhất đối với việc tính toán các di chuyển toán học cũng như các tương tác vật lý của người chơi đối với các đối tượng khác trong trò chơi.
    - myCapsuleCollider2D: thuộc tính nhận thành phần Collider2D, giúp phát hiện các va chạm với các Collider(máy va chạm) khác.
    - myAnimator: thuộc tính nhận thành phần Animator, qua đây thực hiện đặt các giá trị logic tương ứng để thực hiện xử lý các Animation với đối tượng người chơi tương ứng bên trong Unity.
    - hit: thuộc tính kiểu RayCastHit2D, thực hiện phát hiện các Collider xung quanh đối tượng theo chiều vector tới trong hệ trục tọa độ Oxyz. Qua đó ngăn cản các hành vi ngoại lệ như bị kẹt chồng lên đối tượng khác khi thực hiện lướt hay băng qua tường một cách hợp lệ.
    - Các thuộc tính có trong định danh những từ Time, Cooldown, Amount: được sử dụng để tính toán, đưa ra thời gian hồi chiêu hợp lý, thời gian delay giữa các đòn tấn công (tránh những trường hợp tấn công liên hoàn chỉ trong một vài frame, một vài tíc tắc), hoặc cũng có thể là khoảng cách, độ lớn nào đó sẽ được đưa vào tính toán vật lý để thực hiện các hành vi đánh trả hay là lướt.
    - firePoint, laserPrefab, laserSpeed: các thuộc tính liên quan đến đòn tấn công đặc biệt dạng phóng tia laser của người chơi, bao gồm vị trí(Transform), prefab(đối tượng tia laser) và speed(tốc độ của tia laser).
    - isDashButtonDown: trả về giá trị đang ấn nút K (lướt/ né) hay không, kiểu logic
    - player: thuộc tính quan trọng số 1 của lớp, mang kiểu PlayerManager của chính lớp này, nơi mọi thuộc tính của người chơi sẽ được khởi tạo ngay khi scirpt của lớp này được kích hoạt trong Unity, đồng thời được sử dụng toàn khối Assembly để có thể tương tác với việc xử lý các hành vi, sự kiện quan trọng khác.
- Mô tả chi tiết các phương thứ của lớp PlayerManager:
    - Start(): nơi các giá trị, thuộc tính quan trọng như bắt Animation, bắt lớp bao CapsuleCollider2D hay nhận cơ thể vật lý Rigidbody2D sẽ được khởi tạo hoặc gán giá trị tại frame(khung hình) đầu tiên khi script này được kích hoạt trong Unity.

- Update(): cập nhật liên tục mỗi frame của trò chơi, được sử dụng để xử lý, tính toán các thuộc tính được cập nhật theo khung hình như movement, moveDirection, currentTime, …
- FixedUpdate(): cập nhật liên tục mỗi 0.02 giây, không bị ảnh hưởng bởi frame liên tục, được sử dụng để thực hiện, xử lý các tính toán vật lý liên quan đến cơ thể vật lý của đối tượng người chơi.
- OnTriggerEnter2D: xử lý sự kiện đi vào các đối tượng Collider khác có thể trigger, ví dụ như ăn máu, nhặt sách kinh nghiệm, …

- Ngoài ra, lớp PlayerManager còn cài đặt lại các chữ ký phương thúc của interface PlayerAction. Giả thiết rằng trong thời gian tới, trò chơi sẽ có thể có thêm nhiều đối tượng nhân vật mới chứ không chỉ dừng lại ở nhân vật kiếm khách(swordsman) như hiện tại, việc cài đặt một Interface như PlayerAction lên các đối tượng nhân vật khác là tối quan trọng. Bởi, mỗi nhân vật sẽ có một cách xử lý việc thực hiện đòn tấn công thương hay tấn công đặc biệt khác nhau, ví dụ như phù thủy sẽ thực hiện đòn đánh thường là đòn đánh xa có tầm chứ không phải là đòn đánh cận chiến sử dụng kiếm như của nhân vật kiếm khách. Bên cạnh đó, các đối tượng người chơi đều sẽ phải kế thừa gián tiếp các thuộc tính cơ bản của một nhân vật từ lớp PlayerStatistic thông qua lớp thiết lập thông số của đối tượng đó (kế thừa nhiều cấp). Cùng với việc C# là ngôn ngữ chỉ hỗ trợ đơn kế thừa như Java để tránh sự nhập nhằng về việc sử dụng phương thức giữa các lớp cơ sở và dẫn xuất, việc cài đặt interface là sự lựa chọn hợp lý nhất trong tình huống này. Cụ thể về việc cài đặt interfacePlayerAction lên đối tượng người chơi (hiện tại trong trò chơi mới chỉ có kiếm khách) như sau:



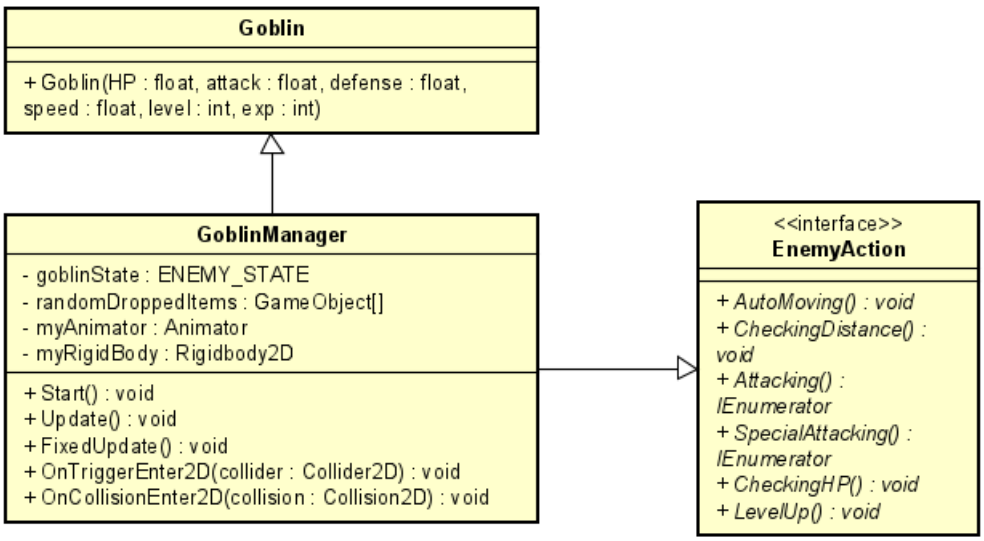
Hình 2.3.2: interface PlayerAction

- MovingPlayer(): di chuyển người chơi thông qua Vector3 (x, y, z) được dựng lên trên hệ trục tọa độ Oxyz. Lý do tại sao game 2D nhưng vẫn sử dụng hệ trục tọa độ 3 chiều Oxyz thay vì hệ trục tọa độ 2 chiều Oxy theo đúng logic sẵn có sẽ được em giải thích chi tiết hơn ở phần công nghệ sử dụng. Ở phương thức này, em sẽ đi vào chi tiết hơn để chỉ ra tính trừu tượng trong Lập trình hướng đối tượng được áp dụng vào project này như thế nào:

```
// MOVING THE PLAYER USING KEYS ON THE KEYBOARD
1 reference
public void MovingPlayer()
{
    // Reset the 3D Vector movement
    movement = new Vector3();

    /*
     * USE A, W, S, D TO MOVE THE PLAYER
     *  -> A: Move Left
     *  -> W: Move Up
     *  -> S: Move Down
     *  -> D: Move Right
     */

    if (Input.GetKey(KeyCode.W))
    {
        movement.y = 1f;            // (x, y) = (0, 1)
    }
    if (Input.GetKey(KeyCode.S))
    {
        movement.y = -1f;           // (x, y) = (0, -1)
    }
    if (Input.GetKey(KeyCode.A))
    {
        movement.x = -1f;           // (x, y) = (-1, 0)
    }
    if (Input.GetKey(KeyCode.D))
    {
        movement.x = 1f;            // (x, y) = (1, 0)
    }

    moveDirection = new Vector3(movement.x, movement.y, 0f).normalized;         // Vector2(x, y) = 1. cos(theta) = x / sqrt(x^2 + y^2)
}
```

- Người chơi chỉ cần biết ấn A là di chuyển trái, D là di chuyển phải chẳng hạn. Tuy vậy, cốt lõi xử lý bên trong phương thức MovingPlayer() thì không hẳn là ai cũng biết được. Cụ thể như việc sử dụng Vector3 được tiêu chuẩn hóa để có thể tính toán được hướng di chuyển của nhân vật một cách chính xá, hay việc ấn nút di chuyển đồng nghĩa với việc gán bộ giá trị (x, y, z) cho hướng di chuyển của người chơi như ấn phím A đồng nghĩa với việc di chuyển sang trái bằng việc cộng vị trí (x, y, z) hiện tại của nhân vật với bộ giá trị (-1, 0, 0) cho nhân vật. Tính trừu tượng ở đây thể hiện qua việc chú trọng đưa ra cách thức thực hiện, ẩn đi những xử lý phức tạp bên trong đối với người chơi.

- AniamtingPlayer(): thiết lập các giá trị liên quan đến các Animation của đối tượng người chơi tương ứng trong Unity.
- Attacking(): xử lý đòn tấn công thường sử dụng kiếm của người chơi, mang kiểm IEnumerator để có thể thực hiện chạy như một trình nền, dừng hay tiếp tục ở khoảng thời gian thích hợp để tương tác với các trạng thái của người chơi.
- SpecialAttacking(): xử lý đòn tấn công đặc biệt sử dụng laser của người chơi.
- LevelUp(): xử lý các vấn đề liên quan đến level(cấp độ) của người chơi trong trò chơi.
- CheckingHP(): kiểm tra lượng máu của người chơi.



Hình 2.3.3: Lớp GoblinManager, Goblin và interface EnemyAction

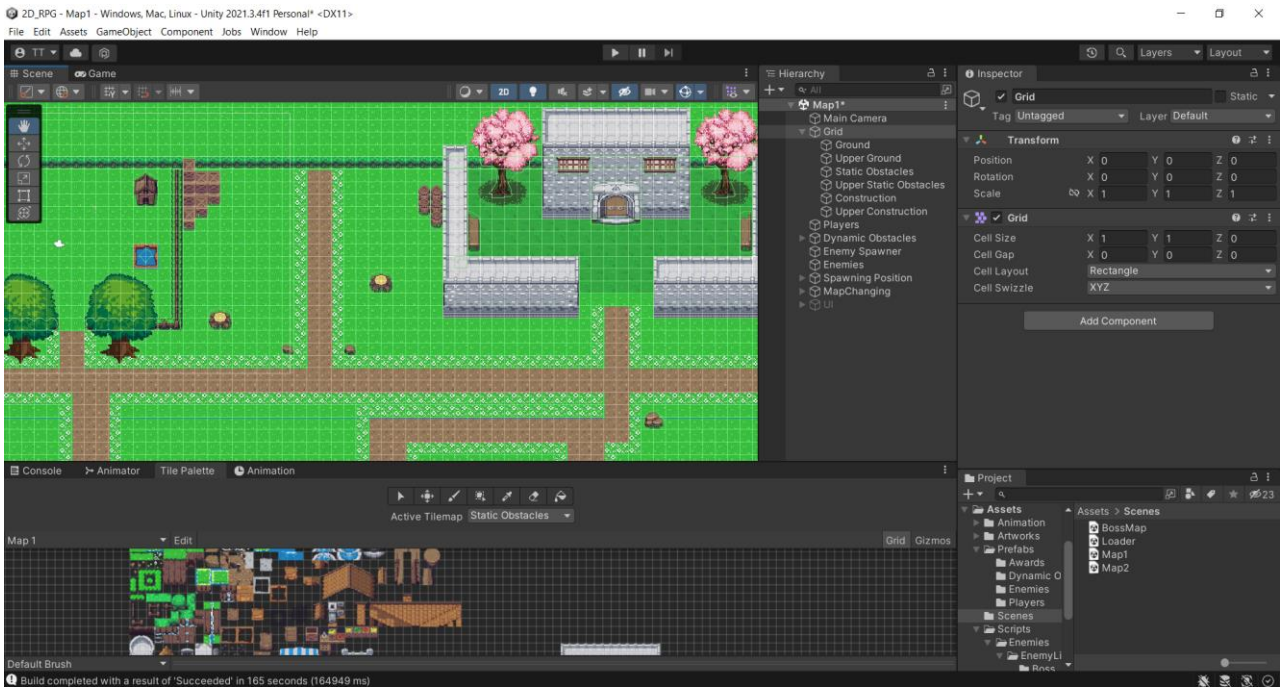
- Tương tự đối với các đối tượng quái vật, việc thể hiện các tính chất của Lập trình hướng đối tượng là rất rõ ràng. Ví dụ như ở đây, lớp quản lý đối tượng quái vật mang tên "Goblin" cũng sẽ kế thừa từ lớp cha thiết lập thông số cho nó là Goblin interface EnemyAction để cài đặt hoàn chỉnh các chữ ký phương thức này. Ngoài Goblin ra còn có WalkableTree hay Boss cũng là các đối tượng quái vật có cách kế thừa giống với con quái Goblin ở đây, có cách xử lý các phương thức tấn công, drop vật phẩm khác nên việc cài đặt interface EnemyAction là hoàn toàn phù hợp.
- Việc các lớp này sẽ được thực thi như thế nào, cũng như giải thích các vấn đề liên quan đến các tính chất của Lập trình hướng đối tượng ở nhóm đối tượng quái vật này gần như tương đồng với nhóm đối tượng nhân vật điều khiển bởi người chơi, đã được giải thích khá chi tiết ở phía trên nên em sẽ không đi sâu vào vấn đề này.

CHƯƠNG 3. CÔNG NGHỆ VÀ THUẬT TOÁN SỬ DỤNG

Hình 3.1: Các công cụ được sử dụng trong việc thiết kế project (C#, Unity, .NET Framework)

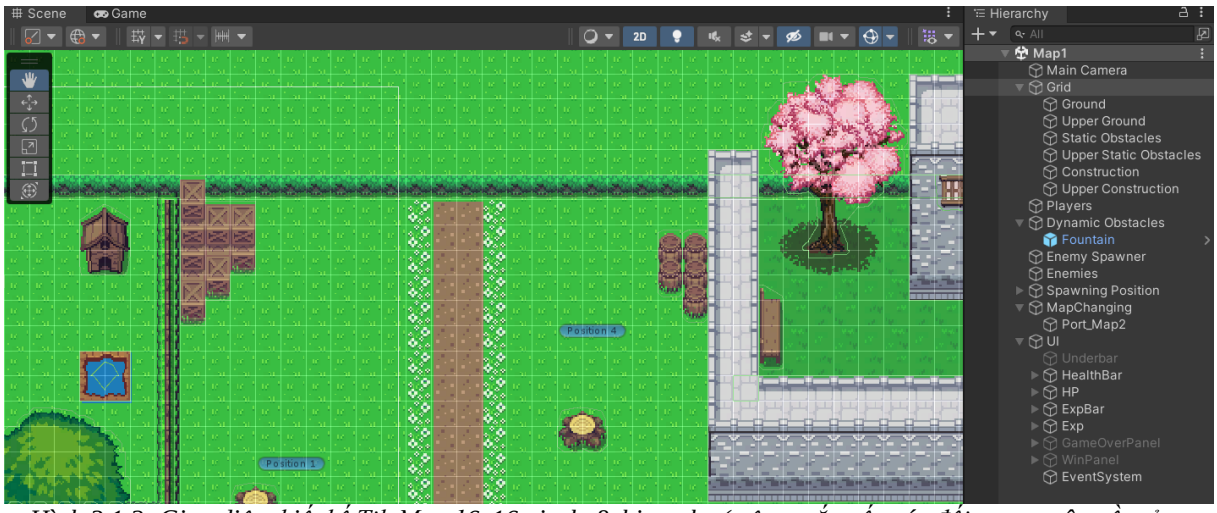
3.1. Unity

Nền tảng Unity là linh hồn trong việc thiết kế lên trò chơi hoàn chỉnh theo những phân tích thiết kế đã được đề ra trong project này.



Hình 3.1.1: Giao diện chính của Unity platform

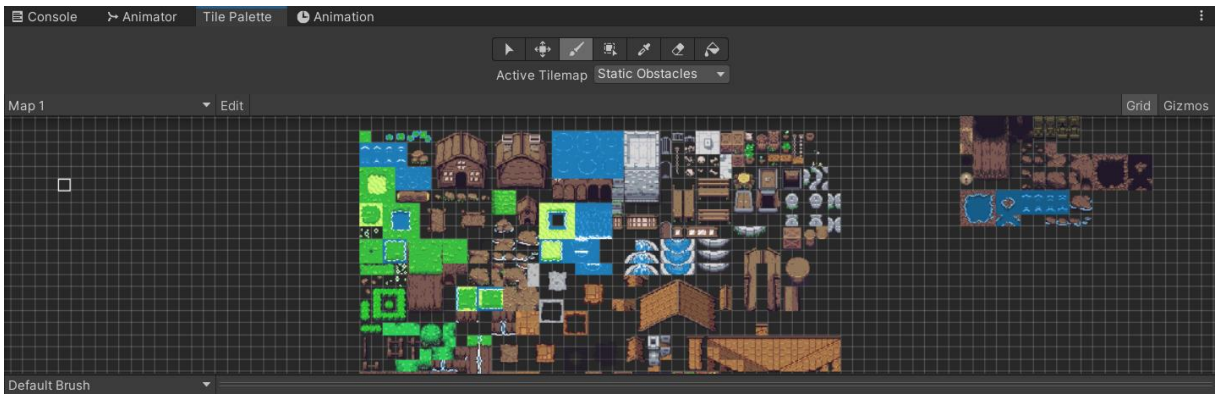
Nền tảng Unity cung cấp những công cụ tuyệt vời cho việc thiết kế, phát triển trò chơi thông qua lượng thư viện tiện ích đồ sộ. Đồng thời, đây cũng là một nền tảng mang tính hướng đối tương mạnh mẽ khi tất cả các thành phần được thiết kế trong trò chơi đều sẽ là thực thi của lớp GameObject, kế thừa từ Object, lớp cơ sở trên cùng của UnityEngine mà mọi đối tượng trong Unity đều tham chiếu tới.



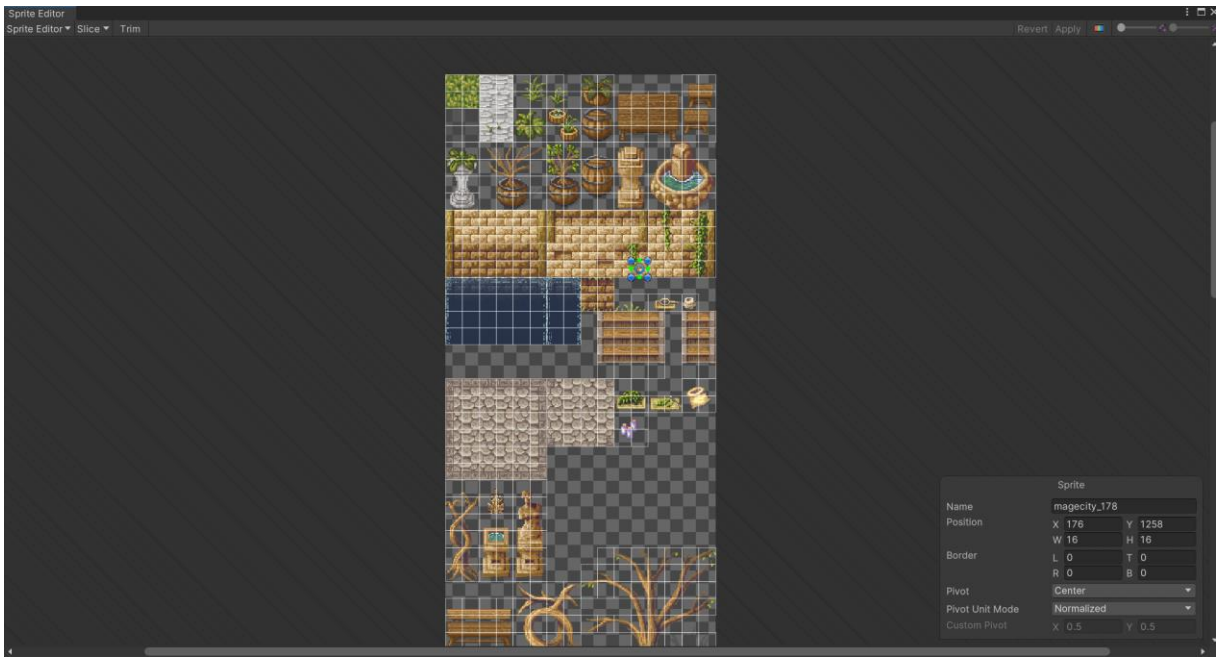
Hình 3.1.2: Giao diện thiết kế TileMap 16x16 pixels & hierachy (trật tự sắp xếp các đối tượng trên nền tảng Unity)

Unity hỗ trợ việc xử lý đồ họa cũng như các hiệu ứng hình ảnh, âm thanh rất nhiều với những công cụ cực kì mạnh mẽ. Cụ thể như:

- Đối tượng thuộc lớp TileMap trong namespace Unity.TileMap hỗ trợ việc lưu các đơn vị đồ họa theo pixels (16x16) qua GridLayout một cách dễ dàng, trực quan. Công cụ Tile Pallete sẽ nhận các Sprite dạng Bitmap được chia nhỏ theo kích cỡ phù hợp với việc thiết kế đồ họa, giao diện của game thông qua Sprite Editor của Unity. Sau đó, đưa chúng vào trong GridLayout của đối tượng TileMap để có thể xử lý một cách tốt nhất.

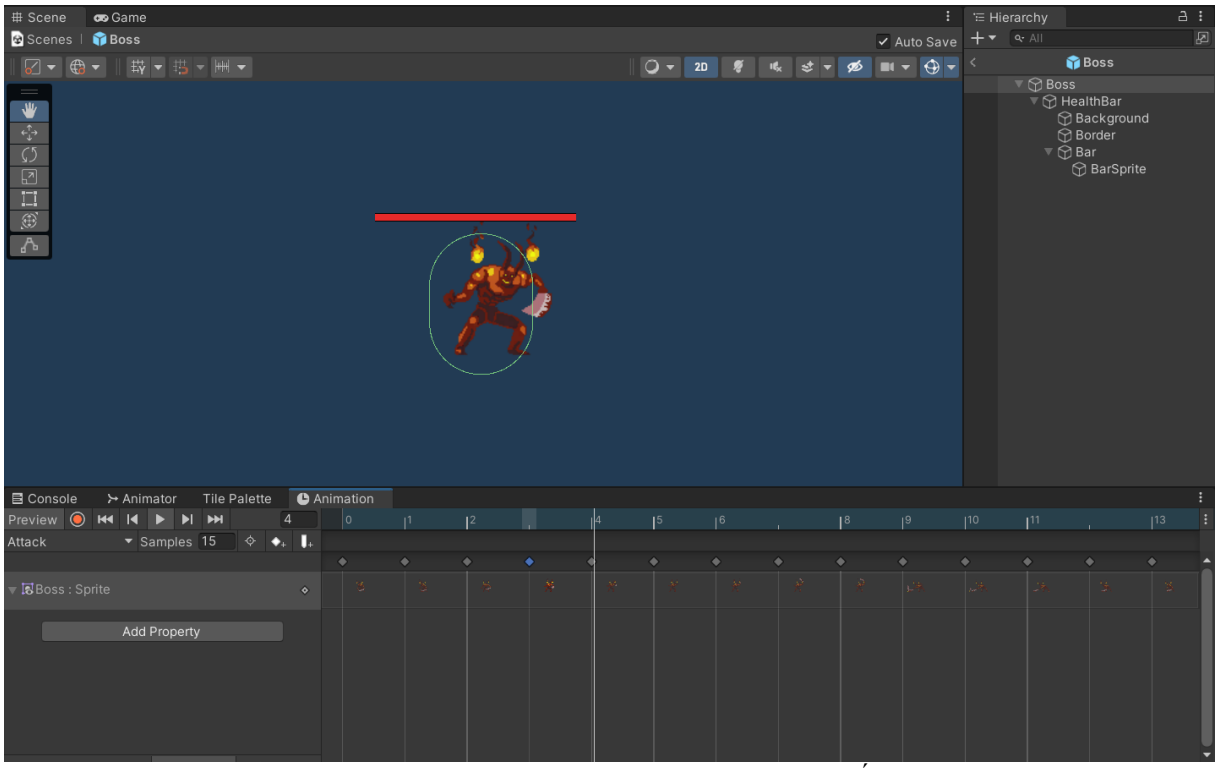


Hình 3.1.3: Công cụ Tile Pallete

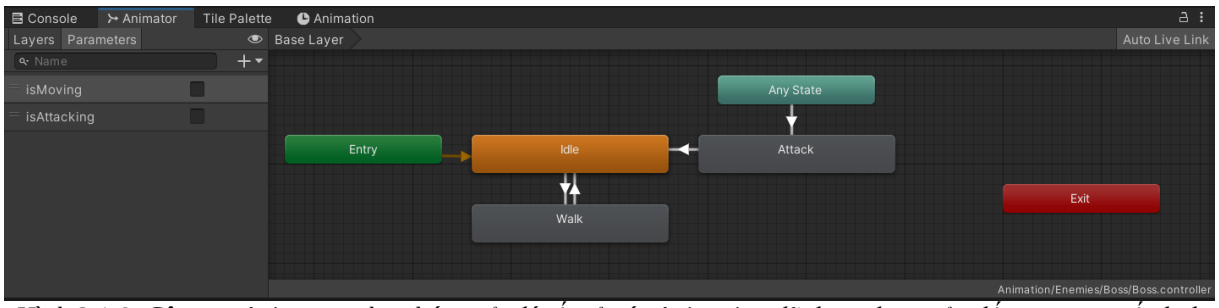


Hình 3.1.4: Công cụ Sprite Editor

- Bên cạnh đó, bộ công cụ dựng animation (Animator, Animation) cũng là một trong những bộ công cụ hỗ trợ cực kỳ mạnh mẽ của nền tảng Unity.



Hình 3.1.5: Công cụ Animation, cho phép dựng hoạt ảnh động của đối tượng trong trò chơi



Hình 3.1.6: Công cụ Animator, cho phép quản lý tất cả các Animation đã được dựng của đối tượng, quyết định trật tự thực hiện chúng phù hợp với yêu cầu của trò chơi

- Một điều tối quan trọng đối với sự vận hành của trò chơi được thiết kế trên nền tảng Unity, đó là việc gắn file script của một lớp nào đó cho đối tượng sẽ thực hiện các hành vi, chức năng tương ứng với lớp đó phía bên trong Unity. Thông qua các phương thức vận hành cơ bản của lớp MonoBehaviour (lớp hành vi cơ bản của tất cả các đối tượng trong Unity), các đối tượng sẽ có thể thực hiện hành vi của mình một cách đúng đắn qua các phương thức dạng thông điệp để thực hiện hành vi cơ bản của lớp MonoBehaviour này:
    - MonoBehaviour.Awake(): khởi tạo ngay khi khối assembly của chương trình bắt đầu thực thi. Được sử dụng để có thể thiết lập các giá trị, thuộc tính mang tính bất biến.
    - MonoBehaviour.Start(): Khởi tạo khi đối tượng có gắn script này được kích hoạt. Được sử dụng để thiết lập các thông số cho đối tượng khi chúng được khởi tạo.
    - MonoBehaviour.Update(): Cập nhật mỗi frame. Dùng để cập nhật các thuộc tính luôn không ngừng thay đổi của đối tượng trong trò chơi (ví dụ như vị trí của nhân vật, quãng đường di chuyển của nhân vật, kiểm tra lượng máu hiện tại của nhân vật, …).
    - MonoBehaviour.LateUpdate(): Cập nhật mỗi frame, được gọi ngay sau khi MonoBehaviour.Update() thực hiện xong. Thường được dùng để có thể điều chỉnh camera hướng tới người chơi một cách liên tục và chính xác.
    - MonoBehaviour.FixedUpdate(): Cập nhật sau mỗi khoảng thời gian nhất định, độc lập với các thông số liên quan đến khung hình/ giây (mặc định trong Unity là 0.02 giây/1 lần gọi FixedUpdate). Ta có thể hiểu đơn giản là đây là phương thức chứa tần số giao động vật lý là f = 1/0.02 = 50 Hz, tức là được gọi một cách cố định 50 lần/giây. Ở trong trò chơi này, ta chỉ cần hiểu đơn giản rằng đây là phương thức sẽ thực hiện xử lý các tính toán, chuyển động vật lý một cách chính xác cho các đối tượng mang thuộc tính cơ thể vật lý(Rigidbody2D) trong trò chơi, ví dụ như tính toán, xử lý động tác lướt/ né, hay cơ chế đánh trả sử dụng xung lực, ...

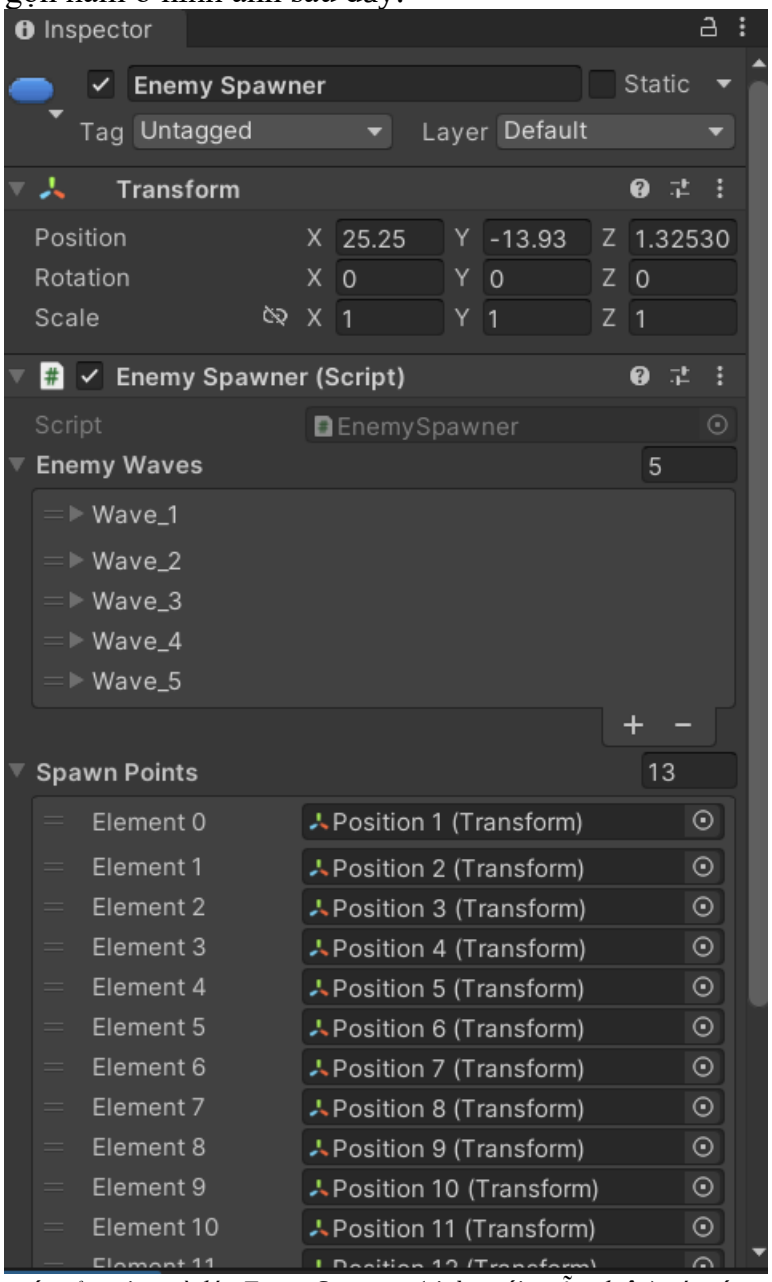
```
public class GameManager : MonoBehaviour
{
    3 references
    public static GameManager instance;
    5 references
    public static GameObject player;

    [SerializeField]
    1 reference
    private GameObject[] players;

    2 references
    private int _charIndex;
    2 references
    public int CharIndex {
        get { return _charIndex; }
        set { _charIndex = value; }
    }

    0 references
    public void Awake() {
        if(instance == null) {
            instance = this;
            DontDestroyOnLoad(gameObject);
        } else {
            Destroy(gameObject);
        }
    }
```

Hình 3.1.8: Lớp GameManager thừa kế từ lớp MonoBehaviour

```
public void Update()
{
    // Calculate the current time using Time.time
    this.currentTime = Time.time;

    // Move the player
    this.MovingPlayer();

    // Attacking
    if (Input.GetKeyDown(KeyCode.J) && currentPlayerState != PLAYER_STATE.Attack)
    {
        StartCoroutine(Attacking());
    }
    else if (Input.GetKeyDown(KeyCode.L) && currentPlayerState != PLAYER_STATE.Attack)
    {
        StartCoroutine(SpecialAttacking());
    }
    else if (currentPlayerState == PLAYER_STATE.Walk)
    {
        this.AnimatingPlayer();
    }

    // Dashing
    if (Input.GetKeyDown(KeyCode.K))
    {
        if ((currentTime - lastDashTime >= dashCooldown || lastDashTime == 0f) && this.currentPlayerState != PLAYER_STATE.Attack)
            isDashButtonDown = true;
    }

    //Updating the level of the player
    this.LevelUp();

    //Checking whether the HP of the player is 0
    this.CheckingHP();
}
```

Hình 3.1.7: Thực thi phương thức Update của đối tượng người chơi qua lớp PlayerManager

Một vấn đề nữa, làm sao để có thể tương tác giữa file script viết bằng C# với Unity, câu trả lời ngắn gọn nằm ở hình ảnh sau đây:



Hình 3.1.8: Tương tác của sricpt từ lớp EnemySpawner (sinh quái ngẫu nhiên) với các vị trí năm trong đối tượng EnemySpawner trong Unity

3.2. C#

C# là một trong những ngôn ngữ lập trình hướng đối tượng phổ biến nhất thời điểm hiện nay. C# được sử dụng để thiết kế ra rất nhiều các nhóm đối tượng sản phẩm khác nhau như web(ASP.NET core), ứng dụng trên Windows(.NET Framework + C# Winform), …

Các thuộc tính cơ bản và quan trọng nhất của Lập trình hướng đối tượng đều được thể hiện một cách rõ nét và mạnh mẽ trong ngôn ngữ lập trình C#

Nền tảng Unity sử dụng ngôn ngữ lập trình C# (Scripting API của Unity được viết bằng C# nên việc nó hỗ trò C# mạnh mẽ nhất là điều hiển nhiên).

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

4 references
public class HealthBar : MonoBehaviour
{
    4 references
    protected float hpPercent;
    4 references
    protected Transform bar;
    2 references
    private float _hpOfGameObject;
    3 references
    public float HPOfGameObject {
        get { return _hpOfGameObject; }
        set { _hpOfGameObject = value; }
    }

    2 references
    private float _maxHpOfGameObject;
    6 references
    public float MaxHPOfGameObject {
        get { return _maxHpOfGameObject; }
        set { _maxHpOfGameObject = value; }
    }

    // Start is called before the first frame update
    4 references
    protected virtual void Start()
    {
        bar = transform.Find("Bar");
        hpPercent = (float) this.HPOfGameObject / (float) this.MaxHPOfGameObject;
        bar.localScale = new Vector3(hpPercent, 1f);
    }
```

Hình 3.2.1: Ngôn ngữ lập trình C# được sử dụng trong việc kết nối với nền tảng Unity

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

1 reference
public interface PlayerAction
{
    1 reference
    public void MovingPlayer();
    1 reference
    public void AnimatingPlayer();
    1 reference
    public IEnumerator Attacking();
    1 reference
    public void LevelUp();
    1 reference
    public void CheckingHP();
    1 reference
    public IEnumerator SpecialAttacking();
}
```

Hình 3.2.2: Interface PlayerAction (các hành vi của người chơi), interface này được lớp PlayerManager cài đặt cụ thể

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

46 references
public class PlayerManager : Player, PlayerAction
{

    // THE TRANSFORM OF THE PLAYER.
    25 references
    private Vector3 movement;
    13 references
    private Vector3 moveDirection;

    // THE CURRENT STATE OF THE PLAYER.
    8 references
    private PLAYER_STATE currentPlayerState = PLAYER_STATE.Walk;

    // GET OTHER COMPONENTS.
    4 references
    private Rigidbody2D myRigidBody;
    9 references
    private CapsuleCollider2D myCapsuleCollider;
    7 references
    private Animator myAnimator;
    0 references
    private RaycastHit2D hit;

    // TIME ATTRIBUTES.
    [SerializeField]
    3 references
    private float currentTime;
    [SerializeField]
    2 references
    private float dashAmount = 5f;
    [SerializeField]
    1 reference
    private float dashCooldown = 3f;
    [SerializeField]
```

Hình 3.2.3: Tính kế thừa, đóng gói được thể hiện rõ nét trong C#

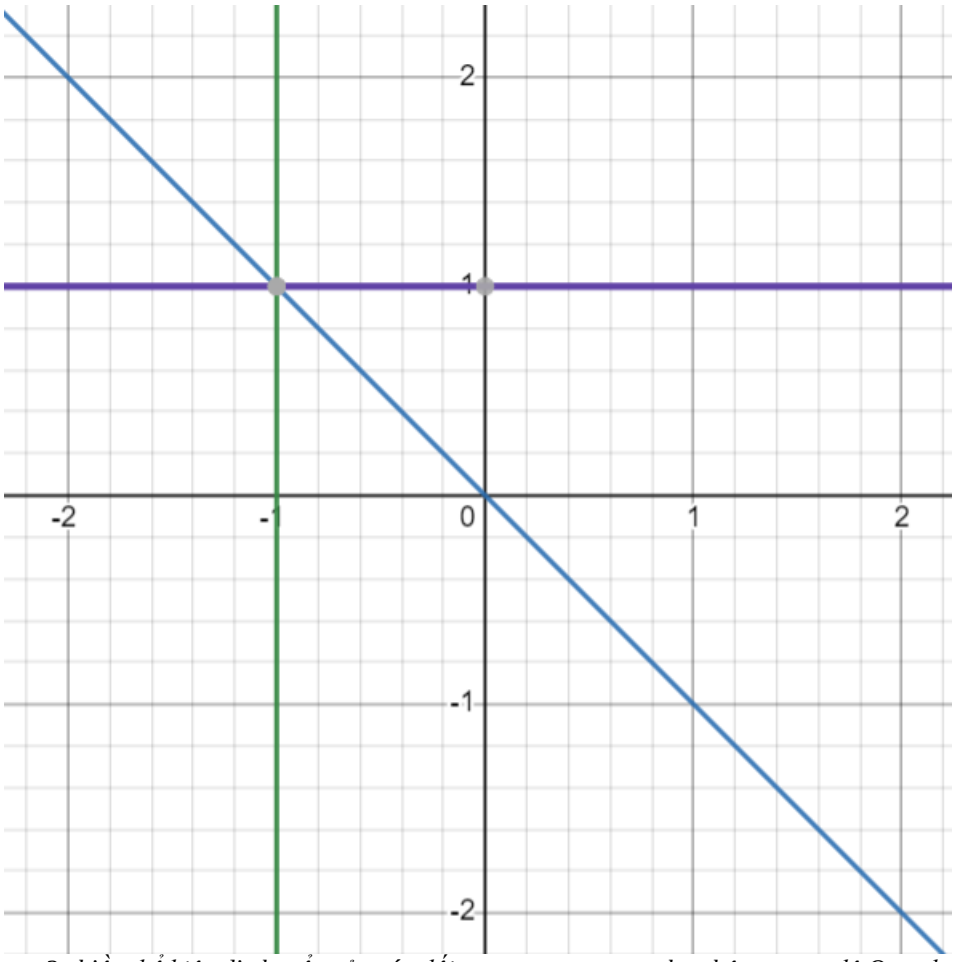
3.3. .NET Framework

Nền tảng .NET Framework hỗ trợ cho việc chạy các ứng dụng C# đa nền tảng. Các thức hoạt động của một chương trình C# trên nền tảng .NET Framework khá đơn giản và có nét tương đồng với các ứng dụng Java chạy trên nền tảng máy ảo Java (JVM - Java Virtual Machine):

- Biên dịch mã nguồn C# sang mã trung gian IL(Intermediate Language).
- Mã IL sau đó được biên dịch bằng một trình biên dịch khác là JIT (Just-In-Time) Compiler, chuyển thành khối mã máy Assembly có tên là CLR (Common Language Runtime) và sẽ được thực thi trên môi trường do .NET Framework quản lý.

Nền tảng Unity ở đây sử dụng/ chạy trên .NET Framework 2.1.

3.4. Unity & một số vấn đề liên quan đến tính toán toán học



Hình 3.4.1: Vector 2 chiều thể hiện di chuyển của các đối tượng trong game theo hệ trục tọa độ Oxy, được tiêu chuẩn hóa với độ dài là 1 (chưa tính vận tốc)

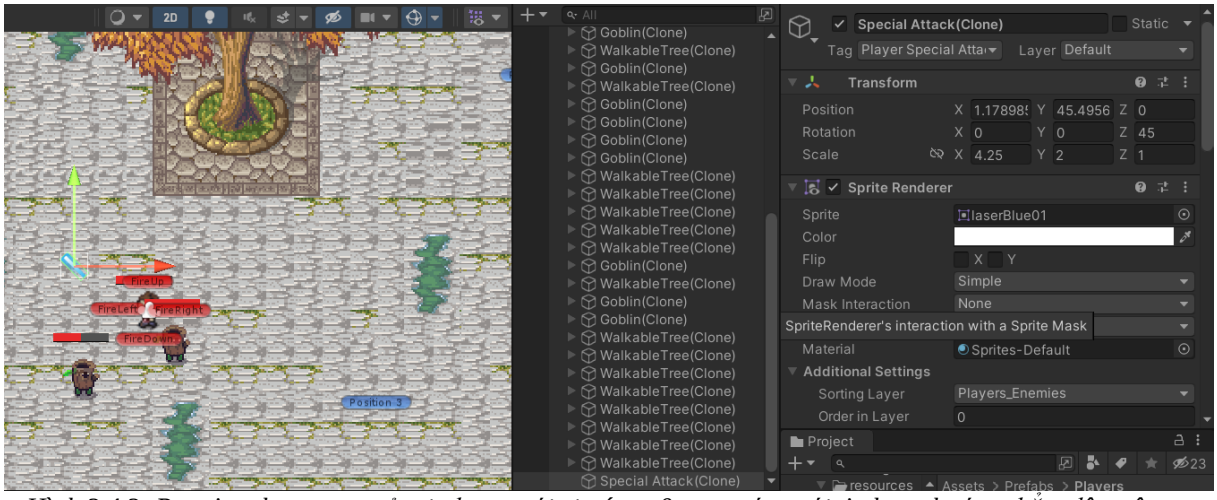
Như đã đề cập ở phần trước, trò chơi sử dụng hệ thống tính toán toán học thông qua hệ trục tọa độ Oxyz để mô phỏng lại các thao tác di chuyển, sắp xếp điểm bitmap, xử lý các vật thể có vận tốc, gia tốc vật lý(người chơi, quái vật, tia laser), … Điều này khiến cho trò chơi trở nên trực quan và dễ tiếp cận hơn dưới góc nhìn toán học.

Trước khi giải thích chi tiết hơn, ta sẽ có một quy ước chung: trục z trong hệ trục tọa độ Oxyz hầu như sẽ không được sử dụng trong thiết kế game 2D. Vậy câu hỏi đặt ra ở đây là tại sao chúng ta vẫn phải cố chấp sử dụng z làm gì khi ta hoàn toàn có thể chuyển về sử dụng Vector2 (x, y) thay cho Vector3 (x, y, z). Ta hãy cùng đến với một phần nhỏ trong đoạn mã nguồn mô tả đòn tấn công đặc biệt sử dụng tia laser:

```
if (this.movement.x == 1 & this.movement.y == 1) {
    firePoint = this.gameObject.transform.Find("FireRight");

    GameObject laser = Instantiate(laserPrefab, firePoint.position, firePoint.rotation);
    laser.transform.localRotation = Quaternion.Euler(0, 0, -45);
    Rigidbody2D rb = laser.GetComponent<Rigidbody2D>();

    Physics2D.IgnoreCollision(myCapsuleCollider, laser.GetComponent<Collider2D>());

    rb.AddForce(moveDirection.normalized * laserSpeed, ForceMode2D.Impulse);
} else if (this.movement.x == 1 & this.movement.y == 0) {
    firePoint = this.gameObject.transform.Find("FireRight");

    GameObject laser = Instantiate(laserPrefab, firePoint.position, firePoint.rotation);
    laser.transform.localRotation = Quaternion.Euler(0, 0, -90);
    Rigidbody2D rb = laser.GetComponent<Rigidbody2D>();

    Physics2D.IgnoreCollision(myCapsuleCollider, laser.GetComponent<Collider2D>());

    rb.AddForce(moveDirection.normalized * laserSpeed, ForceMode2D.Impulse);
} else if (this.movement.x == 1 && this.movement.y == -1) {
    firePoint = this.gameObject.transform.Find("FireRight");

    GameObject laser = Instantiate(laserPrefab, firePoint.position, firePoint.rotation);
    laser.transform.localRotation = Quaternion.Euler(0, 0, -135);
    Rigidbody2D rb = laser.GetComponent<Rigidbody2D>();

    Physics2D.IgnoreCollision(myCapsuleCollider, laser.GetComponent<Collider2D>());

    rb.AddForce(moveDirection.normalized * laserSpeed, ForceMode2D.Impulse);
} else if (this.movement.x == 0 && this.movement.y == -1) {
    firePoint = this.gameObject.transform.Find("FireDown");

    GameObject laser = Instantiate(laserPrefab, firePoint.position, firePoint.rotation);
    laser.transform.localRotation = Quaternion.Euler(0, 0, -180);
```

Hình 3.4.2: Một đoạn mã nhỏ của phương thức SpecialAttack() từ phía người chơi

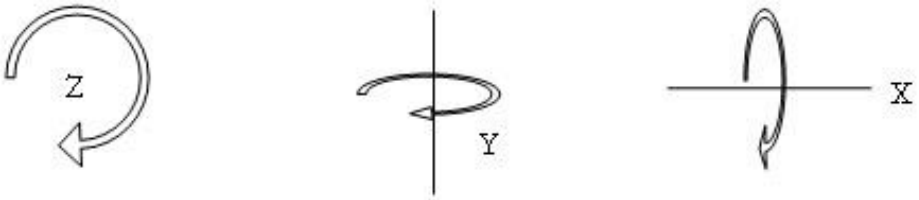
Rõ ràng là game 2D, mô phỏng di chuyển thì cũng chỉ có x với y, vị trí của người chơi thì cũng luôn luôn là transform = Vector3 (x, y, 0), giá trị z = 0 không bao giờ thay đổi như chúng ta đã thấy ở đoạn script của phương thức MovingPlayer(). Nhưng có một vấn đề hiện hữu rõ ràng: để bắn 1 tia laser theo hướng, chúng ta không thể thay đổi giá trị x, y của chúng vì điều đó chỉ giúp thay đổi vị trí của tia laser chứ không hề có một mối liên quan nào đến việc xác đinh hướng của tia laser; đồng thời, việc ta phải vẽ lại tia laser theo 8 hướng tương ứng với mỗi hướng di chuyển của người chơi là lãng phí. Vậy giải pháp ở đây là gì?



Hình 3.4.3: Rotation theo trục z của tia laser, với vị trí z = 0 tương ứng với tia laser hướng thẳng lên trên, ta xoay theo trục z 1 góc 45 độ (-315 độ) để thu được tia laser theo hướng di chuyển A + W

Việc sử dụng Rotation z ở đây đã giúp cho việc điều khiển hướng tia laser theo hướng người chơi trở nên dễ dàng. Hiểu đơn giản rằng trục z là một trục hướng lên thẳng phía trên bản đồ và vuông góc với bản đồ, việc cầm trục z xoay 45 độ sẽ khiến 2

trục x và y xoay theo 1 góc 45 độ, khiến tia laser cũng bị xoay theo 45 độ và có được chiều bay mong muốn.



Hình 3.4.4: Mô tả hướng xoay trong trục tọa độ 3 chiều theo từng trục

```
// MOVING THE PLAYER USING KEYS ON THE KEYBOARD
1 reference
public void MovingPlayer()
{
    // Reset the 3D Vector movement
    movement = new Vector3();

    /*
     * USE A, W, S, D TO MOVE THE PLAYER
     *  -> A: Move Left
     *  -> W: Move Up
     *  -> S: Move Down
     *  -> D: Move Right
     */

    if (Input.GetKey(KeyCode.W))
    {
        movement.y = 1f;            // (x, y) = (0, 1)
    }
    if (Input.GetKey(KeyCode.S))
    {
        movement.y = -1f;           // (x, y) = (0, -1)
    }
    if (Input.GetKey(KeyCode.A))
    {
        movement.x = -1f;           // (x, y) = (-1, 0)
    }
    if (Input.GetKey(KeyCode.D))
    {
        movement.x = 1f;            // (x, y) = (1, 0)
    }

    moveDirection = new Vector3(movement.x, movement.y, 0f).normalized;         // Vector2(x, y) = 1. cos(theta) = x / sqrt(x^2 + y^2)
}
```

Hình 3.4.5: Vẫn là phương thức MovingPlayer(), nhưng tiếp cận dưới góc nhìn toán học

Quay trở lại với một vấn đề đơn giản hơn qua đoạn mã nguồn của hàm MovingPlayer ở hình 3.4.5 trên, ta có thể hiểu đơn giản như sau:

- A: di chuyển trên Oxyz với giá trị Vector3 (-1, 0, 0)
- W: di chuyển trên Oxyz với giá trị Vector3 (0, 1, 0)
- D: di chuyển trên Oxyz với giá trị Vector3 (1, 0, 0)
- S: di chuyển trên Oxyz với giá trị Vector3 (0, -1, 0)

Có một vấn đề ở đây xảy ra, vậy nếu cùng đồng thời ấn A và W thì sẽ là di chuyển tiến tới phá trên, đồng thời sang trái, vậy nếu cùng thay đổi x và y thì rõ ràng, sẽ có hiện tượng ăn gian quãng đường di chuyển xảy ra sau khi đã nhân hướng di chuyển với vận tốc của người chơi, ví du như vận tốc của người chơi là 5f, vậy quãng đường

di chuyển được sẽ là 5*x với x là 1 số lớn hơn 1 và không cố định. Do vậy, ta sử dụng Property Vector3.normalized để luôn đảm bảo dù sử dụng phím di chuyển thế nào, quãng đường di chuyển trong 1 thời gian theo bất kỳ hướng nào của người chơi là như nhau.

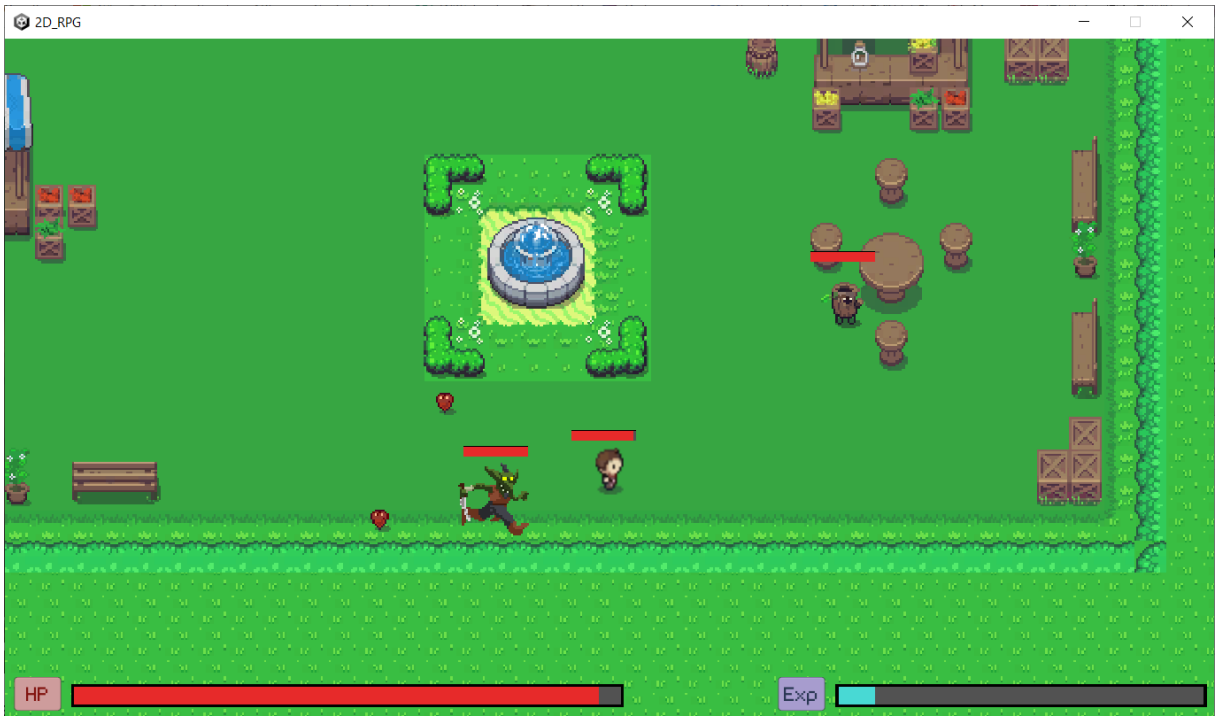
CHƯƠNG 4. XÂY DỰNG CHƯƠNG TRÌNH MINH HỌA

4.1. Kết quả chương trình minh họa

Qua quá trình xây dựng, phân tích thiết kế bài toán, cũng như bắt tay vào thực hiện lập trình C#, xử lý giao diện, đồ họa qua nền tảng Unity, em đã có thể xây dựng một trò chơi nhập vai 2D đơn giản, đáp ứng các yêu cầu một project môn Lập trình hướng đối tượng cần có

Trò chơi đã đáp ứng nhưng mong muốn đề ra từ yêu cầu bài toán:

- Có người chơi, hệ thống quái vật.
- Có các map khác nhau.
- Có hệ thống cấp độ, sức mạnh, cũng như các vật phẩm nâng cấp sức mạnh.
- Có màn đánh boss cuối cùng để quyết định tính thắng thua khi người chơi chơi trò chơi này.



Hình 4.1.1: Giao diện ingame của trò chơi (Map 1)

4.2. Giao diện chương trình

Hình 4.2.1: Giao diện bắt đầu trò chơi

Như đã đề cập ở các phần trước, trò chơi được xây dựng với 2 mức độ chơi là Dễ(Easy) và Khó(Hard)

Hình 4.2.2: Giao diện ingame chính (Map 2)

Giao diện của trò chơi sẽ có thanh máu, thanh kinh nghiệm để người chơi có thể biết được mình đang ở ngưỡng sức mạnh nào trong trò chơi. Khi hạ gục quái vật, các vật phẩm sẽ xuất hiện trên giao diện map hiện tại của người chơi, người chơi hoàn toàn có thể nhận biết và nhặt những vật phẩm có ích này để nâng cao sức mạnh của mình.

Hình 4.2.3: Giao diện ingame chính của màn đánh Boss

Người chơi cần nâng cấp sức mạnh của mình tối đa có thể, chuẩn bị đầy đủ nhất trước khi bước vào màn đánh Boss vì đã vào là không còn đường lui, người chơi sẽ buộc phải chiến đấu với trùm cuối để dành chiến thắng trò chơi.

Hình 4.2.4: Màn hình thua trận

Hình 4.2.5: Màn hình thắng trận

Khi hoàn thành trò chơi, người chơi có thể ấn nút chơi lại (repaly) trên màn hình để bắt đầu chọn lại độ khó và tiếp tục tận hưởng trò chơi.

4.3. Kiểm thử các chức năng đã thực hiện

Tất cả các chức năng đã thực hiện:

- Chọn độ khó để bắt đầu trò chơi.
- Tấn công đối thủ:
  - Tấn công thường bằng kiếm (J).
  - Tấn công đặc biệt bằng tia laser (L).
- Di chuyển (A, W, S, D).
- Lướt/ né một cự ly ngắn (K), không thể dịch chuyển xuyên vật thể hay kẹt vào vật thể khác.
- Chuyển map (đi vào vùng thay đổi map trên bản đồ).
- Đánh boss (map đánh boss, không có cửa trở lại).
- Hiện màn hình thất bại khi máu người chơi xuống 0HP, hiện màn hình chiến thắng khi người chơi đánh bại Boss.
- Bắt đầu lại trò chơi sau khi màn chơi kết thúc (chiến thắng/ thất bại).

⇨ Kết luận: Tất cả các chức năng đều đã hoạt động tốt theo yêu cầu bài toán đặt ra. Không xảy ra lỗi khi thực hiện trò chơi, cũng như không tìm thấy ngoại lệ nào xảy ra.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Thông qua việc xây dựng project trò chơi nhập vai 2D cơ bản, em đã đạt được nhiều điều:

- Nắm rõ các kiến thức về Lập trình hướng đối tượng.
- Làm quen, sử dụng tốt ngôn ngữ lập trình C#, nền tảng phát triển game Unity với cốt lõi là hệ thống game engine UnityEngine.
- Xây dựng được tư duy phân tích, thiết kế bài toán theo hướng đối tượng.
- Có thể làm ra một sản phẩm game mà bản thân luôn mong muốn dù mới chỉ dừng ở mức cơ bản.

Tuy vậy, xuyên suốt quá trình thực hiện project này, cho tới khi sản phẩm được hoàn thành, bản thân em nhận ra mình vẫn còn những điều chưa thực hiện tốt hay chưa đủ khả năng để thực hiện:

- Sử dụng các mẫu đồ họa miễn phí: việc em sử dụng nhiều mẫu đồ họa khác nhau trong trò chơi khiến cho đôi khi, đồ họa của trò chơi không quá ăn khớp với nhau ở một vài vị trí trên bản đồ.
- Các chức năng tấn công của quái vật vẫn còn khá thô sơ, chưa có nhiều đòn tấn công mạnh, bất ngờ, hấp dẫn.
- Chưa đưa được hệ thống inventory(kho vật phẩm) hay xây dựng hệ thống nhân vật đa dạng vào trong trò chơi
- Số lượng map là 3 vẫn là khá ít với một trò chơi nhập vai 2D, đôi lúc sẽ dễ gây nhàm chán cho người chơi

…

Qua những ưu và nhược điểm trên, định hướng phát triển của project này cũng khá rõ ràng:

- Sử dụng nền đồ họa đẹp mắt, sinh động hơn.
- Xây dựng hệ thống vật phẩm, nhân vật, quái vật, bản đồ, hiệu ứng tấn công, gameplay đa dạng, hấp dẫn hơn với người chơi.
- Áp dụng nhiều kỹ thuật làm game hơn thông qua Unity và C# để ngày càng tối ưu trò chơi, mang đến nhiều sự sáng tạo cho trò chơi hơn.

TÀI LIỆU THAM KHẢO

[1] Bài giảng môn Lập trình hướng đối tượng (ThS Nguyễn Mạnh Tuấn – ThS Trịnh Thành Trung).
[2] Unity Scripting API (https://docs.unity3d.com/ScriptReference/index.html)
[3] C# - .NET Documentations (https://docs.microsoft.com/en-us/dotnet/csharp/)