BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

ĐỒ ÁN TỐT NGHIỆP ĐẠI HỌC NGÀNH KHOA HỌC MÁY TÍNH

XÂY DỰNG GAME 2D FOR GRANDPA

GVHD: ThS. Vũ Đức Huy

Sinh viên: Tạ Đăng Khoa

Mã số sinh viên: 2021602992

Hà Nội - 2025

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG BIỂU

LỜI CẢM ƠN

Lời đầu tiên, em xin chân thành cảm ơn quý Thầy, Cô khoa Công nghệ thông tin trường Đại học Công nghiệp đã hướng dẫn, giúp đỡ nhiệt tình trong suốt quá trình học tập bộ môn để có thể hoàn thành đề tài đồ án của em.

Cảm ơn các thầy cô khoa Công Nghệ Thông Tin đã chỉ dạy cho em những kiến thức chuyên ngành vô cùng quý giá. Những kiến thức đó đã hỗ trợ em rất nhiều trong quá trình tự học và nghiên cứu để hoàn thiện đồ án. Nhờ sự chỉ dạy tận tình đó mà em mới có thể áp dụng các kĩ năng và hiểu biết của bản thân để hoàn thành đồ án tốt nghiệp một cách trọn vẹn.

Trong đó, em xin trân trọng cảm ơn thầy **Vũ Đức Huy** đã tận tình chỉ dạy, cầm tay chỉ việc và tạo mọi điều kiện thuận lợi cho em trong suốt quá trình học tập và hoàn thành đồ án tốt nghiệp.

Trong quá trình hoàn thành đồ án tốt nghiệp không thể tránh khỏi những thiếu sót. Kính mong các quý thầy cô chỉ bảo và đóng góp ý kiến để đề tài đồ án của em được hoàn thiện hơn.

Lời cuối cùng em xin chúc các quý thầy, cô và các bạn luôn dồi dào sức khỏe và thành công trong cuộc sống. Xin chúc những điều tốt đẹp nhất sẽ luôn đồng hành cùng mọi người!

Em xin chân thành cảm ơn.

Sinh viên thực hiện

Tạ Đăng Khoa

LỜI MỞ ĐẦU

1. Lý do chọn đề tài

Ngành công nghiệp game giải trí đang ngày càng phát triển mạnh mẽ. Nhằm đáp ứng nhu cầu giải trí của giới trẻ trong thời đại số hiện nay. Đây cũng là lĩnh vực yêu cầu sự đầu tư nghiên cứu và tư duy lập trình như phát triển phần mềm, website. Vì vậy bản thân em đã quyết định triển khai đề tài game của mình: "*For Grandpa*". Đề tài này giúp em thử sức và trang bị cho bản thân những kiến thức và quy trình về lập trình game.

2. Mục tiêu của đề tài

- Vận dụng các kiến thức đã trang bị trong toàn bộ khoá học ở trường và giải quyết đề tài làm game này.

- Có khả năng giải quyết đề tài làm game này bằng máy tính và viết tài liệu báo cáo kỹ thuật và phi kỹ thuật về vấn đề thực hiện một cách hiệu quả.

- Khảo sát, phân tích, tự học, tự nghiên cứu cách xây dựng, thiết kế game và lập trình game 2D trong Unity, sau đó kiểm thử sản phẩm; tác phong làm việc chuyên nghiệp và tuân thủ đạo đức nghề nghiệp.

- Tăng cường khả năng làm việc độc lập.

3. Đối tượng nghiên cứu

- Sử dụng ngôn ngữ lập trình C# và kiến thức về phần mềm Unity để xây dựng, thiết kế game.

- Trong đề tài kết hợp sử dụng các phần mềm chính sau để thực hiện:

  + Phần mềm Unity version 6000.0.41f1.

  + Phần mềm Visual Studio 2022.

4. Phạm vi nghiên cứu

- Phạm vi nội dung: tập trung nghiên cứu cách xây dựng, thiết kế game và lập trình game 2D trong Unity, rồi tiến hành kiểm thử game.

- Phạm vi thời gian: đề tài được thực hiện trong khoảng 9 tuần từ ngày 28/07/2025 đến 29/09/2025.

5. Bố cục đề tài

Ngoài các phần Mở đầu, Kết luận và Tài liệu tham khảo, báo cáo đồ án gồm 3 chương chính sau.

- Chương 1: Lý thuyết chung về thiết kế game.

- Chương 2: Công cụ xây dựng game.

- Chương 3: Xây dựng game 2D For GrandPa.

- Chương 4: Kiểm thử game 2D For GrandPa.

CHƯƠNG 1: LÝ THUYẾT CHUNG VỀ THIẾT KẾ GAME

1.1. Giới thiệu chung

Trò chơi điện tử ngày càng trở nên phổ biến hơn qua nhiều thập kỷ và ngành công nghiệp game đang bùng nổ. Mọi người chơi game để thư giãn, để trở nên giỏi hơn trong các trò chơi mang tính cạnh tranh, hoặc đơn giản chỉ để giết thời gian. Do đó, ngành công nghiệp game đã trở thành một trong những ngành lớn (và thú vị) nhất trên thế giới. Các trò chơi mới được giới thiệu tới người chơi mỗi ngày bởi các công ty game độc lập và các tập đoàn, và tin tức về ngày phát hành game xuất hiện trên mọi mạng xã hội cũng như trong các quảng cáo.

Kết quả là, ngày càng nhiều lập trình viên và nhà phát triển quan tâm đến việc phát triển và thiết kế game, nhưng không phải ai cũng biết bắt đầu từ đâu vì các kỹ năng cần thiết rất nhiều và đa dạng. Hầu hết mọi người cảm thấy việc phát triển game thật đáng sợ bởi số lượng kỹ năng và kiến thức cần có để tạo ra một trò chơi: lập trình, hoạt hình, thiết kế âm thanh, nghệ thuật môi trường… đây là những kiến thức nền tảng để tạo ra game. Không phải ai cũng có đủ các kỹ năng này, và đó là lý do tại sao các game "triple-A" được phát triển bởi những đội ngũ lớn (hoặc studio game) như Ubisoft, Naughty Dog, Square Enix, CD Projekt Red… nhưng điều đó không có nghĩa là một cá nhân không thể tự mình làm nên một trò chơi hay. Có rất nhiều nhà phát triển game độc lập đã tạo ra những trò chơi xuất sắc như Minecraft của Markus "Notch" Persson, Stardew Valley của Eric Barone, Undertale của Toby Fox, …

Vì vậy, trong đồ án này, tôi muốn tạo ra trò chơi đầu tiên của mình bằng cách từng bước một học cách xây dựng một tựa game 2D platformer. Tôi sẽ giới thiệu những kỹ năng cơ bản của phát triển game với ngôn ngữ lập trình C# và một trong những công cụ làm game phổ biến nhất hiện nay: Unity.

Năm 2000: Tháng 3, FPT Software ký hợp đồng OSDC (Trung tâm phát triển phần mềm cho khách hàng) đầu tiên với khách hàng Harvey Nash, tiền thân của G1 (Trung tâm Sản xuất phần mềm số 1). Sau 6 tháng, dự án đầu tiên với OSDC cho khách hàng Proximus (Bỉ) được khởi động với danh sách 9 người chính thức.

1.2. Lịch sử về game trên máy tính

Alan Turing – người truyền cảm hứng cho khoa học máy tính hiện đại và trí tuệ nhân tạo – là người đầu tiên tạo ra một trò chơi máy tính, đó là chương trình chơi cờ vua trên máy tính vào năm 1947. Mục đích của ông không phải để cho mọi người có thể chơi được, mà là để kiểm nghiệm trí tuệ nhân tạo. Mãi đến năm 1952, ông mới thử nghiệm chương trình với một đồng nghiệp và ông đã giả vờ làm "máy tính" (Tristan Donovan, 2010). Trò chơi đầu tiên của Turing đã truyền cảm hứng cho rất nhiều nhà toán học và nhà khoa học máy tính thời đó, khiến họ quyết định tiếp tục công việc của ông.

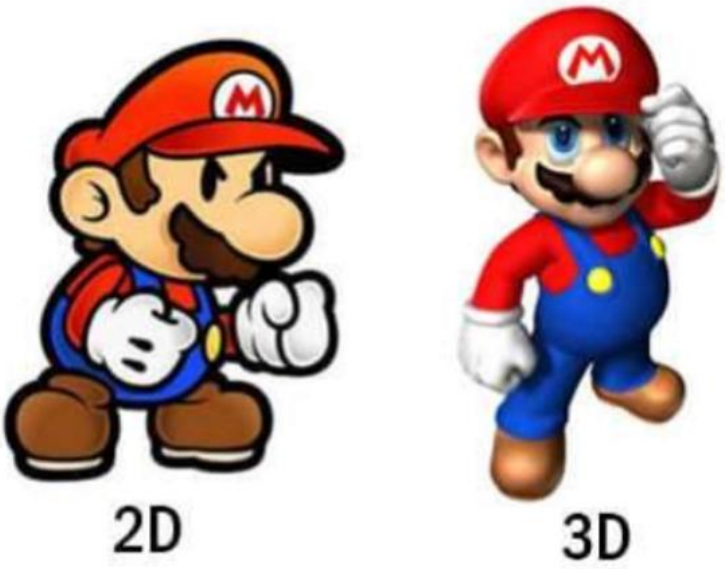
Sau đó vào năm 1972, trò chơi: "Pong" được phát hành trong các máy arcade. Nó trở nên cực kỳ phổ biến đến mức máy arcade bị kẹt do có quá nhiều đồng xu được bỏ vào.

Đến tháng 7 năm 1980, một lát pizza bị thiếu đã truyền cảm hứng cho Toru Iwatani tạo ra: "Pac-Man", trò chơi arcade đầu tiên trở thành hiện tượng mà mọi người có thể chơi tại nhà với máy console của họ.

Tiến nhanh đến đầu thế kỷ 21, chúng ta đã có các kênh tin tức về game, dịch vụ stream dành cho game thủ, cửa hàng game trực tuyến, các sản phẩm ăn theo dựa trên những trò chơi nổi tiếng, cùng với rất nhiều công ty chuyên phát triển game như Electronic Arts, Ubisoft, CD Projekt Red… Với tất cả những điều kể trên, không nghi ngờ gì nữa, ngành công nghiệp game đã trở thành một trong những ngành phát triển nhanh nhất và sinh lợi nhiều nhất trên thế giới.

1.3. Game 2D và game 3D

Thế hệ đầu tiên của trò chơi máy tính là hai chiều (2D) vì công nghệ thời đó còn hạn chế cho việc mô phỏng ba chiều (3D). Nhưng ngày nay, sự phát triển của công nghệ cho phép chúng ta tạo ra các mô phỏng 3D với những công cụ phù hợp. Công nghệ in 3D giờ đã khả thi, các mô phỏng 3D cho phép chúng ta trải nghiệm những điều mà bình thường khó có thể với tới hoặc quá mạo hiểm để thử (như học lái xe, học lái máy bay, mô phỏng hệ mặt trời,…). Vậy thì trò chơi 2D và 3D là gì ?



Hình 1.1: So sánh game 2D và game 3D

Trong trò chơi 2D, thế giới bên trong nó chỉ có hai chiều, nghĩa là nó chỉ bao gồm 2 trục: X và Y. Các trò chơi 2D sử dụng đồ họa phẳng, gọi là sprites, và không có hình học ba chiều. Chúng được vẽ lên màn hình dưới dạng hình ảnh phẳng, và camera (camera trực giao – orthographic) không có phối cảnh (Unity, 2021). Thông thường trong trò chơi 2D, người chơi chỉ có thể di chuyển sang trái và phải, lên và xuống theo trục X-Y vì môi trường trong game 2D được thể hiện phẳng về mặt hình ảnh. Một số ví dụ trò chơi 2D phổ biến: Super Mario Bros, Pac-Man, Donkey Kong,… Hạn chế về chuyển động tuy vậy không có nghĩa là trò chơi 2D kém hấp dẫn hơn trò chơi 3D, điều này phụ thuộc vào sở thích của người chơi. Hơn nữa, đối với những người chơi thích sự giải trí nhẹ nhàng và các game đơn giản, trò chơi 2D là lựa chọn phù hợp hơn vì các game 3D hiện nay

thường đòi hỏi cao hơn và chiếm nhiều dung lượng hơn trên ổ cứng.

Hình 1.2: Ví dụ game 2D Apple Knight Action Platformer

Tuy nhiên, trò chơi 3D mang lại trải nghiệm đắm chìm hơn. Trò chơi 3D thường sử dụng hình học ba chiều, với Vật liệu (Materials) và Kết cấu (Textures) được hiển thị trên bề mặt của các GameObject để khiến chúng trông như môi trường, nhân vật và vật thể rắn tạo nên thế giới game (Unity, 2021). Trong một trò chơi 3D, người chơi có thể di chuyển và nhìn thế giới của nó theo không gian 3D, nghĩa là nó mô phỏng thế giới thực. Các đối tượng trong môi trường 3D có chiều sâu và thể tích vì thế giới game vận hành theo trục X-Y-Z. Người chơi có thể di chuyển lên xuống, trái phải, tiến và lùi. Trò chơi 3D có nhiều khả năng và mục tiêu hơn so với trò chơi 2D.

Hình 1.3: Ví dụ game 3D Genshin Impact

Trong trò chơi 2D, nhân vật chỉ di chuyển trong hai chiều, điều này dễ kiểm soát hơn. Trong khi đó, trò chơi 3D tạo ra nhiều thách thức hơn cho người chơi vì họ phải làm quen với chuyển động 3D trong game, chẳng hạn như dùng phím WASD để di chuyển, phím Space để nhảy và chuột để quan sát xung quanh.

Lý do tôi chỉ muốn viết về phát triển trò chơi 2D là vì nó cơ bản nhất và dễ học nhất. Sau khi học cách làm trò chơi 2D, tôi sẽ có kiến thức, kinh nghiệm và nền tảng để tạo game, nhờ đó việc chuyển sang phát triển game 3D sẽ dễ dàng hơn nếu muốn.

1.4. Game Engine

Hình 1.4: Ví dụ một vài game engine

Game engine là một phần mềm được thiết kế đặc biệt để tạo ra trò chơi điện tử. Các nhà phát triển sử dụng game engine để tạo trò chơi cho nhiều nền tảng khác nhau, chẳng hạn như máy tính, console và điện thoại di động. Nó tồn tại để trừu tượng hóa (đôi khi phụ thuộc nền tảng) các chi tiết trong những tác vụ chung liên quan đến game, như kết xuất (rendering), vật lý và nhập liệu, từ đó giúp các nhà phát triển (nghệ sĩ, nhà thiết kế, lập trình viên kịch bản và thậm chí cả các lập trình viên khác) có thể tập trung vào những chi tiết làm cho trò chơi của họ trở nên độc đáo (Jeff Ward, 2008).

Các thành phần chính của một game engine bao gồm: bộ kết xuất (renderer) cho thiết kế 2D và 3D, bộ xử lý vật lý (physics engine), chức năng kịch bản (scripting), bổ sung âm thanh và các công cụ hoạt hình. Nhiều game engine còn cung cấp các thành phần có thể tái sử dụng như mô hình nhân vật, âm thanh cơ bản, thế giới mẫu. Vì lý do đó, game engine trở thành công cụ mạnh mẽ và tiện lợi để thiết kế hầu như bất cứ thứ gì, không chỉ riêng trò chơi.

Vậy tại sao cần thêm các công cụ thiết kế khác cho hoạt hình và âm nhạc? Bởi vì những công cụ đó được tạo ra chuyên nghiệp và chuyên biệt cho những chức năng riêng. Ví dụ, phần mềm tạo nhạc không cần thành phần hoạt hình, và phần mềm vẽ cũng không nhất thiết phải có thiết kế âm thanh. Việc sử dụng công cụ được thiết kế riêng cho một mục đích sẽ thuận tiện hơn cho các chuyên gia. Do đó, phần mềm thiết kế game, hay game engine, được tạo ra để phục vụ các yếu tố trong phát triển trò chơi.

1.4.1 Các game engine phổ biến

1.4.1.1 Unreal Engine

Hình 1.5: Logo Unreal Engine

Nhiều game hạng nặng (triple-A) được phát triển bằng Unreal Engine nhờ khả năng đồ họa mạnh mẽ trong xử lý ánh sáng, shader, v.v. Engine này cũng là mã nguồn mở, nên liên tục được cộng đồng cải tiến. Unreal Engine phù hợp nhất để tạo ra game 3D và VR nhờ vào các tùy chọn đồ họa. Do chứa các thành phần đồ họa nặng, engine này yêu cầu máy tính mạnh hơn để chạy so với Unity. Ngoài ra, nhiều nhà phát triển còn khẳng định rằng Unreal Engine thích hợp hơn cho các dự án lớn mà bạn có ý định làm việc theo nhóm.

Một số game tiêu biểu: Ark: Survival Evolved, Final Fantasy VII Remake, Gears 5…

1.4.1.2 Godot

Hình 1.6: Logo Godot

Game engine Godot còn khá mới, được phát hành vào năm 2014 và chỉ mới đây mới trở nên phổ biến. Nó miễn phí và mã nguồn mở, rất phù hợp cho những lập trình viên mới bắt đầu. Godot có thể dùng để phát triển cả game 2D và 3D nhưng khá nhẹ. So với Unity hoặc Unreal Engine, chất lượng sản phẩm không bằng. Godot không có sức mạnh đồ họa như Unreal Engine, cũng không có cộng đồng hỗ trợ mạnh như Unity, nhưng lại dễ học và dễ sử dụng nếu bạn muốn bắt đầu hành trình phát triển game. Về nền tảng, đa số các trò chơi phát triển bằng Godot là game mobile.

Một số game tiêu biểu: Sigil Breakers, Kip, The Kingdom of Avalae…

1.4.1.3 Unity

Hình 1.7: Logo Unity

Unity đã trở thành lựa chọn phổ biến cho các nhà phát triển game indie trên toàn thế giới kể từ năm 2005. Nó phù hợp không chỉ cho phát triển game 2D và 3D mà còn rất mạnh trong thiết kế thực tế ảo (VR) và thực tế tăng cường (AR). Engine này được cập nhật hàng năm với nhiều nội dung mới, và nhờ cộng đồng đông đảo, Unity có một cửa hàng tài nguyên (asset store) chứa rất nhiều tài sản miễn phí và trả phí. Unity hoàn toàn miễn phí, giúp bất kỳ ai cũng có thể bắt đầu học và phát triển game dễ dàng.

Một số game tiêu biểu: Cuphead, Inside, Ori and the Blind Forest…

Lý do tôi chọn Unity Engine trong đồ án này là vì nó phù hợp để tự mình phát triển game. Thông thường, để làm một game cần cả một nhóm vì có quá nhiều thành phần mà một người khó có thể đảm nhận hết, hoặc phải mất rất nhiều thời gian để thành thạo tất cả các kỹ năng cần thiết (hoạt hình, âm thanh, thiết kế môi trường, lập trình…). Unity cung cấp sẵn một số thành phần miễn phí, nhờ đó chúng ta có thể tập trung vào lập trình

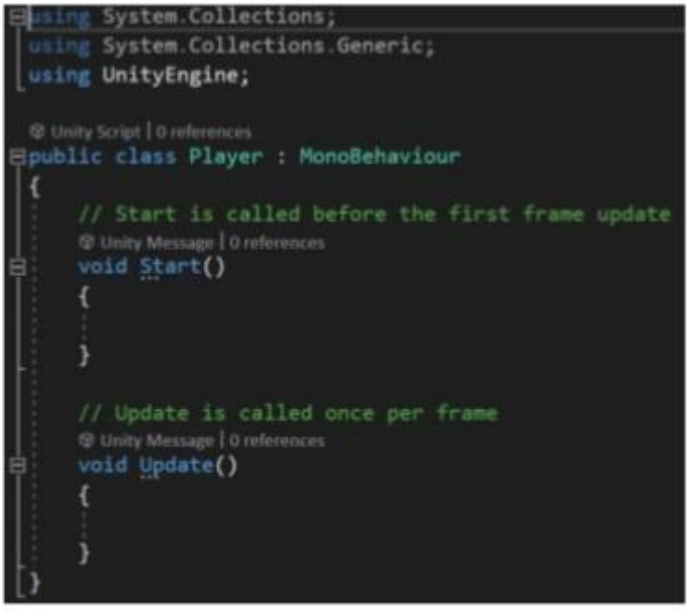
CHƯƠNG 2: CÔNG CỤ XÂY DỰNG GAME

2.1. Ngôn ngữ lập trình C#

C# (đọc là C-sharp), được lấy cảm hứng từ nhiều tính năng của F#, là một ngôn ngữ lập trình lần đầu ra mắt công chúng vào năm 2000. Đây là ngôn ngữ đa mục đích được thiết kế để phát triển ứng dụng trên Windows và cần .NET Framework để hoạt động. C# có thể dùng để lập trình gần như bất kỳ thứ gì, nhưng đặc biệt tốt cho việc phát triển ứng dụng Windows và trò chơi. Một số lập trình viên web cũng chọn C# để xây dựng ứng dụng web, và nó cũng khá phổ biến cho phát triển ứng dụng di động.

C# dễ học, tương đối dễ đọc, nhưng do tính linh hoạt, nó khó để nắm vững toàn bộ những gì mà C# mang lại. Đây là một ngôn ngữ phức tạp, việc làm chủ nó sẽ mất nhiều thời gian hơn so với các ngôn ngữ đơn giản như Python hoặc HTML; người học cần nắm một lượng mã đáng kể để tạo ra chương trình nâng cao. Tuy vậy, thời gian bỏ ra để học là rất xứng đáng. C# là một kỹ năng đang được nhiều công ty công nghệ săn đón. Trên thị trường việc làm, mỗi ngày đều có vô số ứng dụng cần nhà phát triển có kiến thức và kinh nghiệm tốt về C#. Vì vậy, những ai học C# sẽ có cơ hội nghề nghiệp rộng mở.

Tôi chọn C# vì tính linh hoạt của nó. Tôi có thể học C# không chỉ để làm game mà còn để có cơ hội tốt hơn trong hành trình trở thành lập trình viên trong bất kỳ lĩnh vực nào của thế giới Công nghệ thông tin.



Hình 2.1: Ví dụ một tập lệnh C#

Đây là một ví dụ về tập lệnh (script) C#. Cấu trúc của C# rất dễ hiểu: chúng ta có một lớp (class), và bên trong nó có các phương thức (methods) hoặc hàm về những gì lớp này có thể làm. Một lớp giống như một bản thiết kế (blueprint) cho một đối tượng. Trong đời thực, một đối tượng có hình dạng, màu sắc và các chức năng; điều này cũng tương tự với một lớp trong C#. Bên trong lớp này, chúng ta tạo ra các phương thức, khai báo các giá trị và thuộc tính. Do đó, một lớp định nghĩa loại chức năng và dữ liệu mà các đối tượng của nó có.

Trong ví dụ về lớp "Player", chúng ta có các phương thức "Start" và "Update". Các phương thức này đã có sẵn các chức năng được mô tả ở trên chúng trong các dòng chú thích màu xanh lá cây. Chúng ta có thể tự tạo các phương thức mới và đặt tên phù hợp với chức năng của chúng (ví dụ: PlayerMove, PlayerDie, PlayerIdle...). Tên phương thức không được có khoảng trắng ở giữa. Bên trong phương thức, chúng ta bắt đầu bằng dấu ngoặc nhọn mở "{" và kết thúc bằng dấu ngoặc nhọn đóng "}".

Chú thích (Comments) trong C# không có bất kỳ chức năng nào trên mã. Chúng ta sử dụng chú thích đơn giản để giải thích mã của mình và làm cho nó dễ đọc hơn. Một dòng chú thích đơn bắt đầu bằng hai dấu gạch chéo "//".

Ba dòng chữ "using" dùng để gọi không gian tên (namespace). Đây là một cách để chúng ta tiết kiệm thời gian viết tên không gian tên mỗi khi cần sử dụng một phương thức trong không gian tên đó. Ví dụ, thông thường chúng ta phải viết: " System.Console.WriteLine("Hello World!");" để in dòng chữ "Hello World!", nhưng với không gian tên using System;, chúng ta có thể viết Console.WriteLine("Hello World!"); để viết tắt.

2.2. Tổng quan về Engine Unity

2.2.1 Giới thiệu

Hình 2.2: Unity Engine

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

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

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

Sức mạnh: Unity có thể tạo ra được nhiều loại game 2D, 3D đa dạng, dễ sử dụng với người làm game chưa chuyên nghiệp, chất lượng cao, chạy hầu hết trên các hệ điều hành.

Hình 2.3: Game 2D Among Us được làm bởi Unity Engine

Hình 2.4: Game 3D Sons Of The Forest được làm bởi Unity Engine

Sự tiện lợi: nếu chúng ta là một người chuyên dùng 3Dmax, hay Maya hoặc phần mềm mã nguồn mở Blender thì quả là thật tuyệt, chúng ta sẽ có một lợi thế lớn khi viết game trên Unity này, bởi công việc tạo các mô hình 2D, 3D sẽ trở lên dễ dàng hơn rất nhiều, việc kết hợp giữa người lập trình và người thiết kế các mô hình sẽ nhanh và hiệu quả hơn. Trong Unity chúng ta có thể import trực tiếp các file mô hình đang thiết kế và sẽ thiết kế hoàn thiện tiếp nếu chưa xong trong khi đó công việc import chỉ diễn ra một lần. Không như việc phải dùng các công cụ khác để thực hiện viết game chúng ta sẽ phải xuất chúng ra một dạng nào đó và

mỗi lần sửa lại phần mô hình chúng ta lại phải import lại, và như thế là quá mất thời gian trong việc tạo và chỉnh sửa các mô hình theo ý muốn. Ngoài ra Unity còn cho chúng ta trực tiếp tạo các mô hình nếu muốn. Việc đặt các thuộc tính vật lý trong Unity cũng cực kỳ dễ dàng và hỗ trợ sẵn nhiều chức năng.

2.2.2 Sơ lược lịch sử hình thành và phát triển của Unity

Phần lõi của Unity ban đầu được viết bởi Joachim Ante vào năm 2001. Sau đó công ty được hình thành vào năm 2005 và bắt đầu với phiên bản 1.0.

Đến năm 2007, Unity được nâng lên phiên bản 2.0. Unity bắt đầu hỗ trợ iPhone vào năm 2008. Vào tháng 6/2010, Unity chính thức hỗ trợ Android và cho ra đời phiên bản 3.0 có hỗ trợ Android vào tháng 9/2010 và bây giờ là phiên bản Unity 6. Có thể thấy tốc độ phát triển của Unity khá nhanh.

Unity được trên 1.2 triệu người đăng ký sử dụng gồm Bigpoint, Cartoon Network, Coca-Cola, Disney, Electronic Arts, LEGO, Microsoft, NASA, Ubisoft, Warner Bros, các hãng phim lớn nhỏ, các chuyên gia độc lập, sinh viên và những người đam mê.

2.2.3 Tính năng của Engine Unity

Môi trường phát triển được tích hợp với tính năng kế thừa, khả năng chỉnh sửa đồ họa, chức năng kiểm tra chi tiết, và đặc biệt tính năng xem trước game ngay trong lúc xây dựng (live game preview).

Triển khai được trên nhiều nền tảng:

- Chương trình độc lập trên Windows và Mac OS.
- Trên web, thông qua Unity Web Player plugin cho Internet Explorer, Firefox, Safari, Opera, Chrome, cho cả Windows và Mac OS.
- Trên Mac OS Dashboard widget.
- Cho Nintendo Wii (cần mua license thêm.)
- Cho iPhone, iPad application (cần mua license thêm.) Cho Google Android (cần mua license thêm.)
- Cho Microsoft Xbox 360 (cần mua license thêm.) Cho Sony PlayStation 3 (cần mua license thêm.)

Tài nguyên (model, âm thanh, hình ảnh, ...) được tải vào trong Unity và tự động cập nhật nếu tài nguyên có sự thay đổi. Unity hỗ trợ các kiểu định dạng từ 3DS Max, Maya, Blender, Cinema 4D và Cheetah3D.

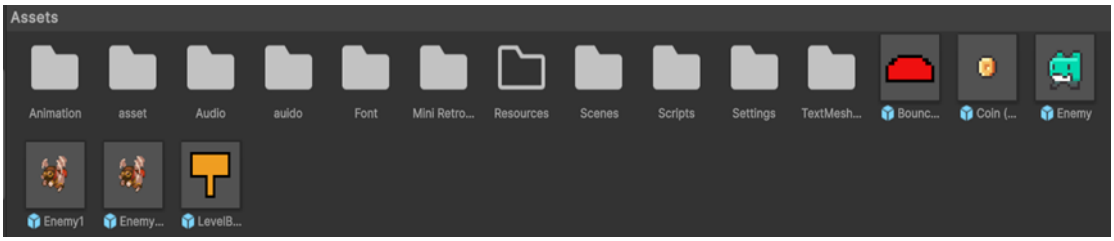
Graphics engine sử dụng Direct3D (Windows), OpenGL (Mac, Windows), OpenGL ES (iPhone OS), và các API khác trên Wii.

Hỗ trợ bump mapping, reflection mapping, parallax mapping, Screen Space Ambient Occlusion v...v...

Unity Asset Server: Đây là một tính năng khá mới của Unity, theo đó Unity sẽ cung cấp một hệ thống quản lý theo dạng phiên bản cho tất cả asset và cả script. Đây là một kho chứa các tài nguyên cần thiết cho việc làm game. Khi import cũng như sửa chữa, trạng thái của asset ngay lập tức được cập nhật. Server chạy trên database opensource PostgreSQL và có thể truy cập trên cả Mac lẫn Windows, Linux. Asset Server.

2.3. Các thành phần trong Unity

2.3.1 Assets



Hình 2.5: Hình ảnh về Assets

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

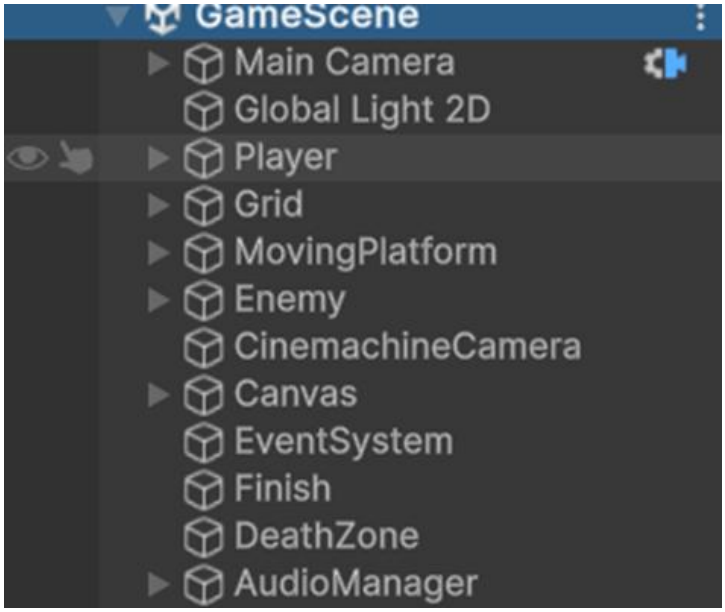
Các asset do chính những nhà phát triển game tạo ra và có thể được download miễn phí hoặc trả phí trên Unity Asset Store. Đây là một trong những tính năng rất hay của Unity. Các asset này sẽ giúp giảm thiểu rất nhiều thời gian cho việc thiết kế và lập trình game.

2.3.2 Scenes

Hình 2.6: Hình ảnh về Scenes

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

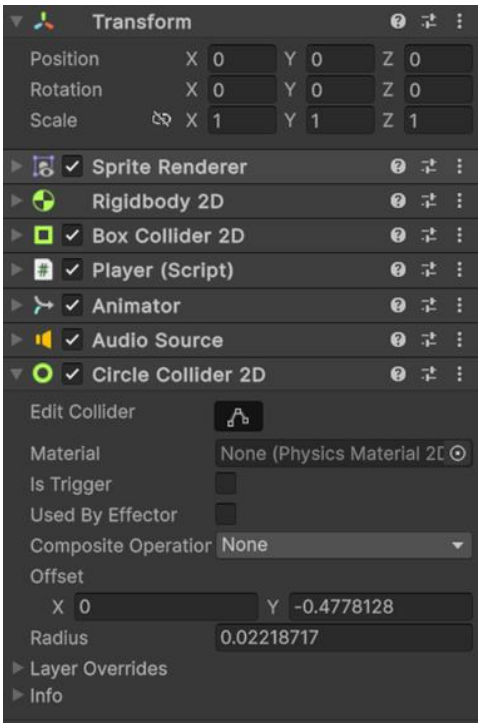
2.3.3 GameObjects



Hình 2.7: Hình ảnh về GameObjects

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

2.3.4 Components



Hình 2.8: Hình ảnh về Components

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

2.3.5 Scripts

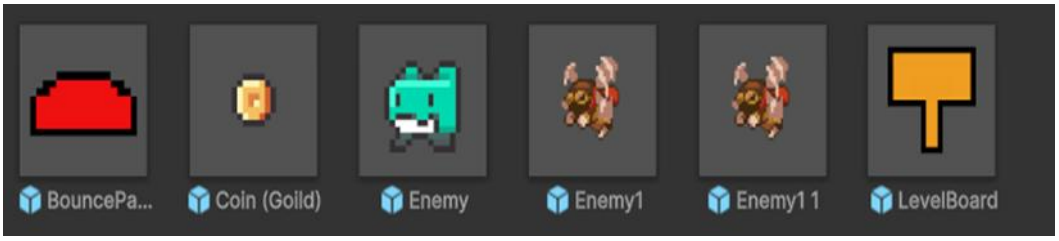
Hình 2.9: Hình ảnh về Scripts

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

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

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

2.3.6 Prefabs



Hình 2.10: Hình ảnh về Prefabs

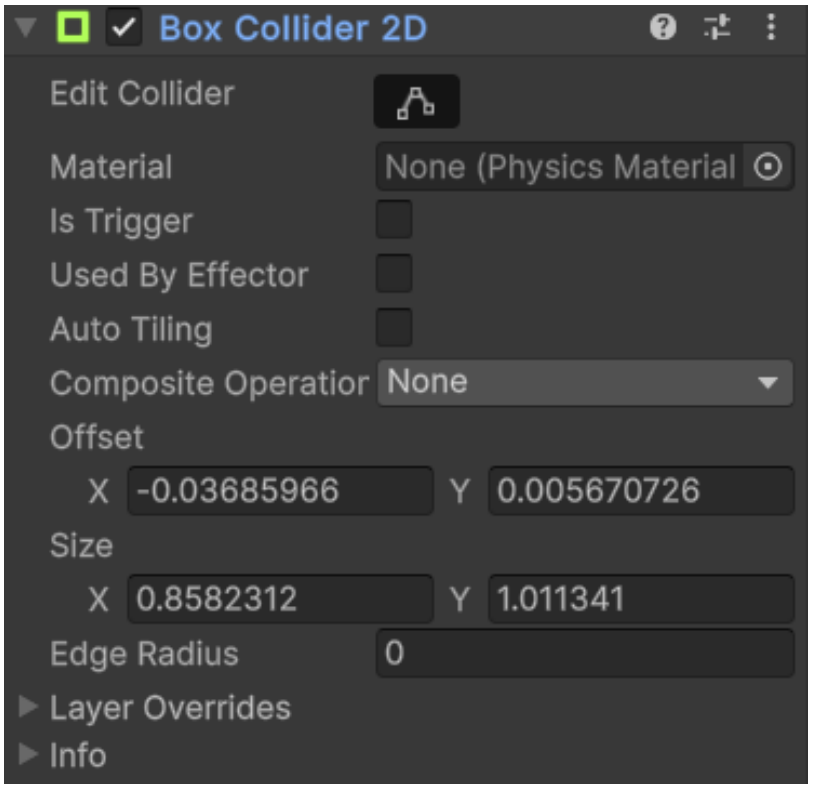
Prefabs thực chất là Game Object được lưu trữ lại để tái sử dụng. Các Game Object được nhân bản từ một prefab sẽ giống nhau hoàn toàn, ngoại trừ thành phần Transform để phân biệt và quản lý được tốt hơn.

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

2.3.7 Collider

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

- Box collider: Va chạm cho các vật hình hợp chữ nhật đối với 3D và hình chữ nhật đối với 2D.



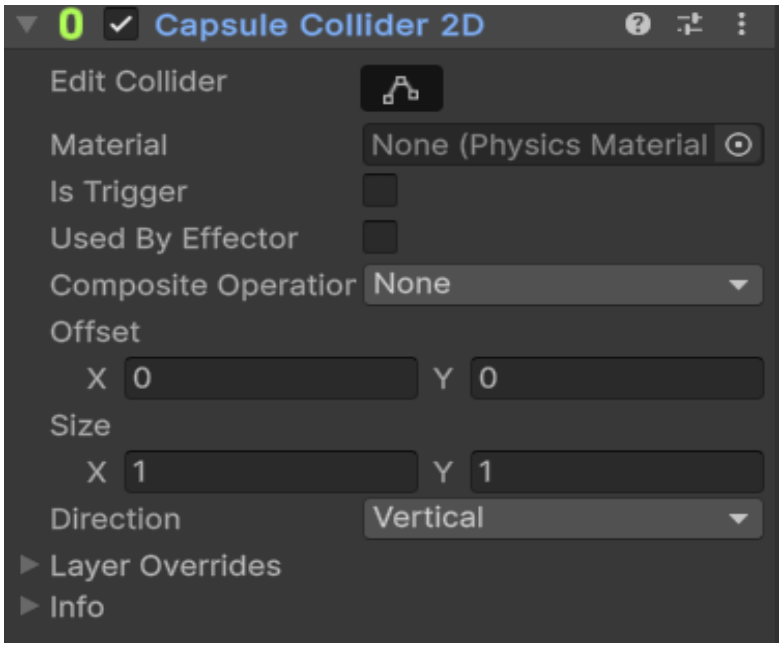
Hình 2.11: Hình ảnh về Box Collider 2D

+ Edit Collider: chỉnh sửa kích thước và vị trí của collider trực tiếp trong Scene View. Các điểm điều khiển sẽ xuất hiện trên viền của box collider để bạn kéo và thay đổi.
+ Material: Cho phép bạn gán một Physics Material 2D (vật liệu vật lý 2D) vào collider này. Vật liệu vật lý có thể thay đổi cách các vật thể tương tác với nhau
+ Is Trigger: dạng true/false, cho phép va chạm có đi xuyên qua không?
+ Used By Effector: Bật tùy chọn này để collider có thể tương tác với các Effector 2D (bộ hiệu ứng), chẳng hạn như Platform Effector 2D (tạo nền tảng cho phép vật thể đi xuyên qua một chiều) hoặc Surface Effector 2D (tạo lực đẩy trên bề mặt).
+ Auto Tiling: Tùy chọn này thường được sử dụng cho các tilemap. Khi bật, collider sẽ tự động thay đổi hình dạng để khớp với các ô tile mà nó được áp dụng.
+ Composite Operation: Tùy chọn này chỉ xuất hiện khi collider được sử dụng với một Composite Collider 2D.

Nó cho phép bạn xác định cách các collider riêng lẻ kết hợp lại với nhau.

+ Offset: Điều chỉnh vị trí của box collider so với tâm của GameObject.
    - X: Dịch chuyển collider theo trục ngang.
    - Y: Dịch chuyển collider theo trục dọc.
+ Size: Điều chỉnh kích thước của box collider.
    - X: Chiều rộng của box.
    - Y: Chiều cao của box.
+ Edge Radius: Đây là một tùy chọn hữu ích để làm tròn các góc của box collider. Điều này có thể giúp làm mượt chuyển động của các vật thể khi va chạm vào các cạnh, tránh bị kẹt.

- Capsule Collider: Va chạm dành cho các khối hình trụ.



Hình 2.12: Hình ảnh về Capsule Collider 2D

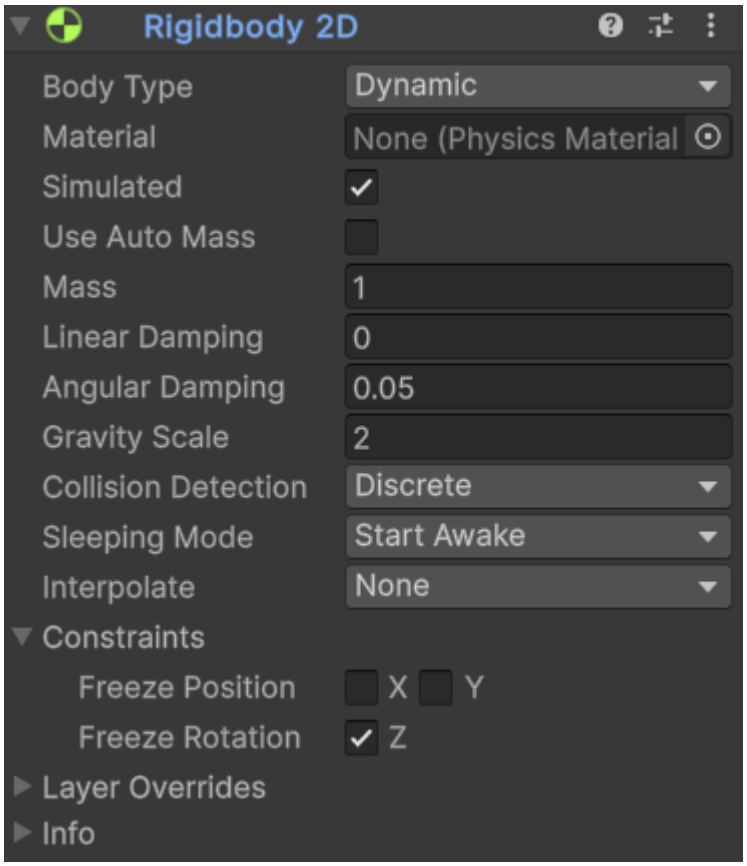
+ Edit Collider: Khi nhấp vào biểu tượng này, bạn có thể chỉnh sửa kích thước và vị trí của collider trực tiếp trong Scene View. Bạn có thể kéo các điểm điều khiển trên viền để thay đổi hình dạng của nó.
+ Material: Dùng để gán một Physics Material 2D. Vật liệu vật lý này sẽ xác định các thuộc tính như ma sát (friction) và độ nảy (bounciness) của collider.

+ Is Trigger: Khi được chọn, collider sẽ hoạt động như một "vùng kích hoạt" thay vì là một vật thể va chạm cứng. Nó sẽ không ngăn cản các vật thể khác đi qua, mà thay vào đó sẽ gửi các sự kiện như OnTriggerEnter2D, rất hữu ích cho việc tạo ra các vùng nhặt đồ, cổng dịch chuyển, hoặc các vùng kiểm tra.
+ Used By Effector: Tùy chọn này cho phép collider tương tác với các Effector 2D như Platform Effector 2D hoặc Surface Effector 2D, cho phép bạn tạo ra các hiệu ứng vật lý phức tạp hơn trên bề mặt.
+ Composite Operation: Chỉ xuất hiện khi bạn sử dụng một Composite Collider 2D. Nó giúp bạn kết hợp nhiều collider thành một collider duy nhất để tối ưu hóa hiệu suất và làm cho vật lý ổn định hơn.
+ Offset: Điều chỉnh vị trí của collider so với tâm của GameObject.
    - X: Dịch chuyển theo trục ngang.
    - Y: Dịch chuyển theo trục dọc.
+ Size: Điều chỉnh kích thước của hình viên thuốc.
    - X: Chiều rộng.
    - Y: Chiều cao.
+ Direction: Xác định hướng của capsule collider.
    - Vertical: Hướng theo chiều dọc (chiều cao lớn hơn).
    - Horizontal: Hướng theo chiều ngang (chiều rộng lớn hơn).

- Mesh Colilider: Va chạm dạng lưới dành các vật thể không xác định hình thể.
- Sphere Colider: va chạm áp dụng cho các vật hình cầu.
- Wheel collider: dành cho vật thể hình bánh xe. Nó sẽ mô phỏng hệ thống va chạm giống với các bánh xe.
- Terrain Collider: dành cho các vật thể địa hình và terrain collider sẽ dựa theo địa hình đó

2.3.8 Rigidbody

Hệ thống mô phỏng vật lí trong game. Cũng chia ra 2D.



Hình 2.13: Hình ảnh về Rigidbody 2D

- Body Type : Đây là một trong những thuộc tính quan trọng nhất, xác định cách game object tương tác với hệ thống vật lý.
    + Dynamic: Loại mặc định, phù hợp nhất cho các đối tượng di chuyển, chịu tác động của tất cả các lực (trọng lực, va chạm, script). Ví dụ: nhân vật, viên đạn, các vật thể có thể di chuyển.
    + Kinematic: Đối tượng này chỉ di chuyển khi bạn thay đổi vị trí hoặc xoay chúng bằng script. Chúng không chịu tác động của trọng lực hay va chạm từ các đối tượng Dynamic, nhưng vẫn có thể ảnh hưởng đến các đối tượng khác. Thích hợp cho các đối tượng di chuyển theo kịch bản như nền tảng di động hoặc cửa.
    + Static: Dành cho các đối tượng không bao giờ di chuyển trong game. Chúng không chịu tác động của bất kỳ lực

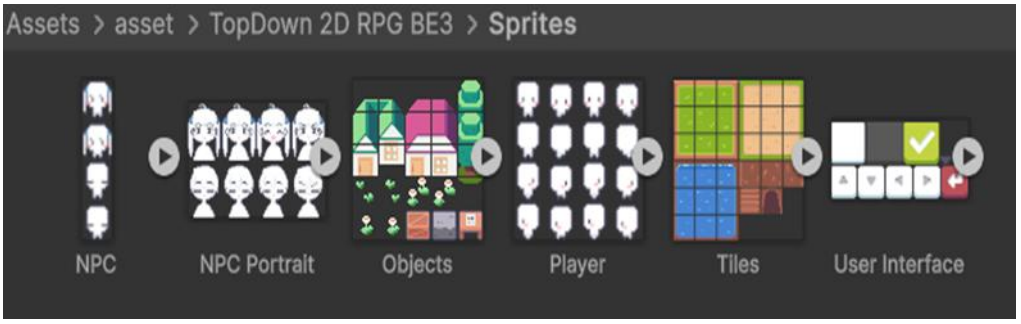
nào và tối ưu hóa hiệu suất. Ví dụ: mặt đất, tường, các vật thể cố định.

- Material (Vật liệu): Tương tự như collider, bạn có thể gán một Physics Material 2D để điều chỉnh các đặc tính vật lý như ma sát (friction) và độ nảy (bounciness).
- Simulated: Khi được bật, Rigidbody sẽ được mô phỏng bởi công cụ vật lý của Unity. Nếu tắt, nó sẽ bị vô hiệu hóa vật lý.
- Use Auto Mass (Sử dụng Khối lượng Tự động): Khi được bật, Unity sẽ tự động tính toán khối lượng của Rigidbody dựa trên các collider được gắn vào nó. Nếu tắt, bạn có thể tự đặt giá trị Mass.
- Mass (Khối lượng): Khối lượng của đối tượng. Khối lượng càng lớn, đối tượng càng khó bị tác động bởi các lực và va chạm.
- Linear Damping (Giảm xóc Tuyến tính): Hệ số làm giảm tốc độ di chuyển của đối tượng theo thời gian. Giá trị 0 không có ma sát không khí, giá trị cao hơn sẽ làm đối tượng chậm lại nhanh hơn.
- Angular Damping (Giảm xóc Góc): Tương tự như Linear Damping, nhưng làm giảm tốc độ xoay của đối tượng.
- Gravity Scale (Hệ số Trọng lực): Nhân với trọng lực toàn cục của game để tạo ra trọng lực riêng cho đối tượng. Giá trị 1 là trọng lực bình thường, 0 không có trọng lực, và giá trị âm sẽ làm vật thể bay lên.
- Collision Detection (Phát hiện Va chạm):
    + Discrete (Rời rạc): Mặc định, phát hiện va chạm tại các bước vật lý riêng lẻ. Hiệu quả nhưng có thể bỏ sót va chạm nếu đối tượng di chuyển quá nhanh.
    + Continuous (Liên tục): Phát hiện va chạm chính xác hơn bằng cách kiểm tra va chạm trên toàn bộ quãng đường di chuyển. Phù hợp cho các vật thể nhỏ và di chuyển nhanh để tránh "xuyên" qua các vật thể khác, nhưng tốn hiệu suất hơn.
- Sleeping Mode (Chế độ Ngủ):
    + Never Sleep: Rigidbody không bao giờ "ngủ" và luôn được mô phỏng, ngay cả khi nó đứng yên.

+ Start Awake: Mặc định, Rigidbody bắt đầu ở trạng thái "thức" và sẽ "ngủ" (dừng mô phỏng để tiết kiệm hiệu suất) khi nó đứng yên trong một thời gian.
+ Start Asleep: Rigidbody bắt đầu ở trạng thái "ngủ" và chỉ "thức dậy" khi bị tác động bởi một lực.

- Interpolate (Nội suy): Giúp làm mượt chuyển động của đối tượng.
  + None: Không có nội suy. Chuyển động có thể bị giật ở tốc độ khung hình thấp.
  + Interpolate: Nội suy vị trí của frame trước để làm mượt chuyển động.
  + Extrapolate: Nội suy vị trí của frame tiếp theo.
- Constraints: Ràng buộc những hạn chế về chuyển động của Rigidbody.
  + Freeze Position (Đóng băng Vị trí): Ngăn Rigidbody di chuyển theo các trục đã chọn.
    - X: Không di chuyển theo trục ngang.
    - Y: Không di chuyển theo trục dọc.
  + Freeze Rotation (Đóng băng Xoay): Ngăn Rigidbody xoay.
    - Z: Thường được dùng cho game 2D để ngăn nhân vật hoặc vật thể xoay khi va chạm.

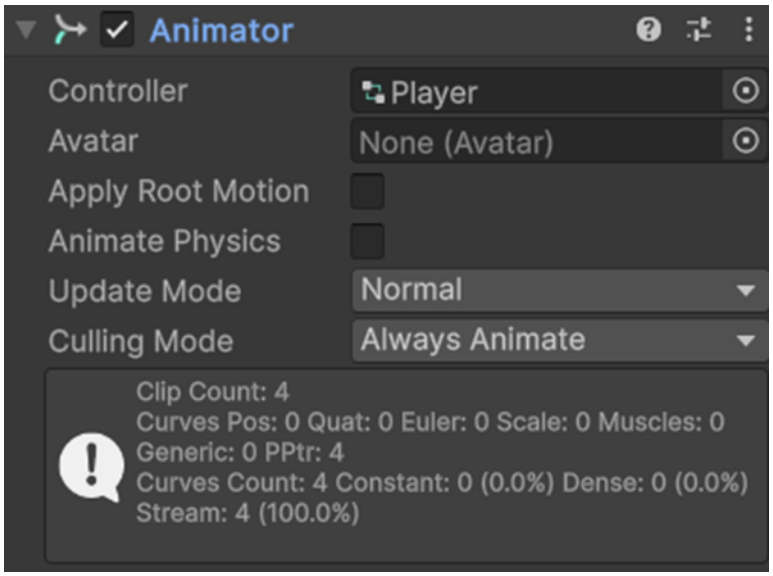
2.3.9 Sprite



Hình 2.14: Hình ảnh về Sprite

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

2.3.10 Animator



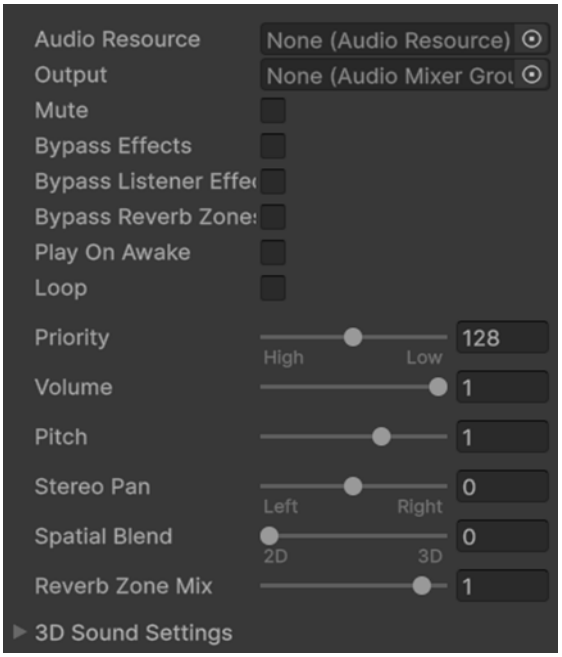
Hình 2.15: Hình ảnh về Animator

Trong 2D thì animation là tập một hình ảnh động dựa trên sự thay đổi liên tục của nhiều sprite khác nhau. Trong 3D là một tập hợp các sự thay đổi theo thời gian của đối tượng trong không gian. Mỗi thay đổi là một key frame. Key Frame hay Frame là một trạng thái của một animation. Aminator gồm các thành phần:

- Controller: Đây là thuộc tính quan trọng nhất. Bạn phải gán một Animator Controller vào đây. Animator Controller giống như một bản đồ quy trình, chứa các trạng thái hoạt ảnh (animation states), các chuyển đổi (transitions) giữa chúng và các tham số (parameters) để điều khiển chúng. Nó là bộ não giúp bạn lập trình logic cho các hoạt ảnh.
- Apply Root Motion: Khi được bật, các hoạt ảnh có thể tự điều khiển vị trí và xoay của toàn bộ game object (còn gọi là "root"). Ví dụ, một hoạt ảnh chạy có thể di chuyển nhân vật tiến về phía trước thay vì chỉ thay đổi hình ảnh. Nếu tắt, bạn sẽ phải tự điều khiển chuyển động của nhân vật bằng script.
- Animate Physics: Tùy chọn này rất hữu ích khi bạn đang làm việc với các Rigidbody. Khi bật, các cập nhật hoạt ảnh sẽ được thực hiện trong quá trình cập nhật vật lý (FixedUpdate) thay vì cập nhật khung hình bình thường (Update). Điều này giúp các hoạt ảnh khớp với vật lý hơn và ngăn ngừa các vấn đề giật lag hoặc va chạm không chính xác.

- Update Mode: Xác định thời điểm các hoạt ảnh được cập nhật.
    + Normal: Cập nhật hoạt ảnh trong mỗi khung hình (frame update). Đây là chế độ mặc định và phổ biến nhất.
    + Animate Physics: Cập nhật trong FixedUpdate (cùng lúc với vật lý), tốt cho các hoạt ảnh tương tác với Rigidbody.
    + Unscaled Time: Cập nhật hoạt ảnh bằng thời gian thực, không bị ảnh hưởng bởi Time.timeScale. Hữu ích cho các hoạt ảnh UI hoặc hiệu ứng trong màn hình tạm dừng game.
- Culling Mode: Xác định khi nào Unity ngừng cập nhật hoạt ảnh để tối ưu hóa hiệu suất.
    + Always Animate: Hoạt ảnh luôn được cập nhật, bất kể game object có hiển thị trên màn hình hay không.
    + Cull Update Transforms: Ngừng cập nhật hoạt ảnh khi game object không hiển thị trên màn hình.
    + Cull Completely: Ngừng cập nhật toàn bộ component Animator khi game object không hiển thị.

2.3.11 Audio Source



Hình 2.16: Hình ảnh về Audio Source

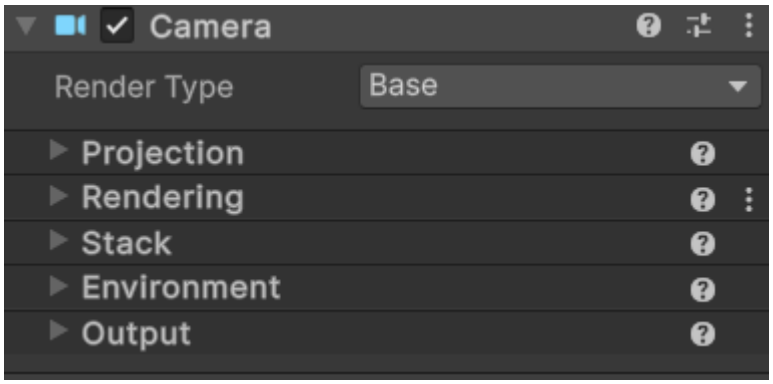
Âm thanh trong game. Gồm cả âm thanh trong môi trường 2D và 3D:

- Audio Resource: Đây là nơi bạn gán một tệp âm thanh (ví dụ: .wav hoặc .mp3) vào. Khi bạn gán một tệp, Audio Source sẽ phát tệp âm thanh đó.
- Output: Cho phép bạn gán Audio Mixer Group. Audio Mixer giúp quản lý các nhóm âm thanh, cho phép bạn điều chỉnh âm lượng, thêm hiệu ứng (effects), và định tuyến âm thanh một cách tập trung.
- Mute: Khi bật, Audio Source sẽ bị tắt tiếng, nhưng vẫn tiếp tục phát. Hữu ích cho việc tạm thời tắt âm thanh mà không làm gián đoạn việc phát.
- Bypass Effects: Khi bật, âm thanh từ nguồn này sẽ bỏ qua tất cả các hiệu ứng (effects) được áp dụng trong Audio Mixer.
- Bypass Listener Effects: Bỏ qua các hiệu ứng được áp dụng cho Audio Listener.
- Bypass Reverb Zones: Bỏ qua các hiệu ứng hồi âm (reverb) được tạo bởi Reverb Zone.
- Play On Awake: Khi bật, Audio Source sẽ tự động phát âm thanh khi game object được kích hoạt lần đầu.
- Loop: Khi bật, âm thanh sẽ tự động lặp lại từ đầu khi nó kết thúc. Thường dùng cho nhạc nền hoặc âm thanh môi trường.
- Priority (Ưu tiên): Xác định tầm quan trọng của nguồn âm thanh này. Khi có quá nhiều nguồn âm thanh cùng lúc, Unity sẽ ưu tiên phát các nguồn có giá trị thấp hơn (ưu tiên cao hơn).
- Volume: Điều chỉnh âm lượng của nguồn âm thanh. Giá trị 1 là âm lượng tối đa.
- Pitch (Cao độ): Điều chỉnh cao độ của âm thanh. Giá trị 1 là cao độ gốc. Tăng giá trị sẽ làm âm thanh nhanh và cao hơn; giảm giá trị sẽ làm âm thanh chậm và trầm hơn.
- Stereo Pan: Điều chỉnh cân bằng âm thanh giữa loa trái (trái) và loa phải (phải) cho âm thanh 2D.
- Spatial Blend: Đây là một thanh trượt quan trọng để xác định âm thanh là trong môi trường 2D hay 3D.

+ 2D (Giá trị 0): Âm thanh không bị ảnh hưởng bởi vị trí của người nghe (Audio Listener). Tốt cho nhạc nền hoặc hiệu ứng UI.
+ 3D (Giá trị 1): Âm thanh phát ra từ một vị trí trong không gian 3D. Âm lượng sẽ thay đổi tùy thuộc vào khoảng cách và hướng của người nghe.

- Reverb Zone Mix: Điều chỉnh mức độ ảnh hưởng của hiệu ứng hồi âm từ Reverb Zone lên âm thanh này.

2.3.12 Camera



Hình 2.17: Hình ảnh về Camera

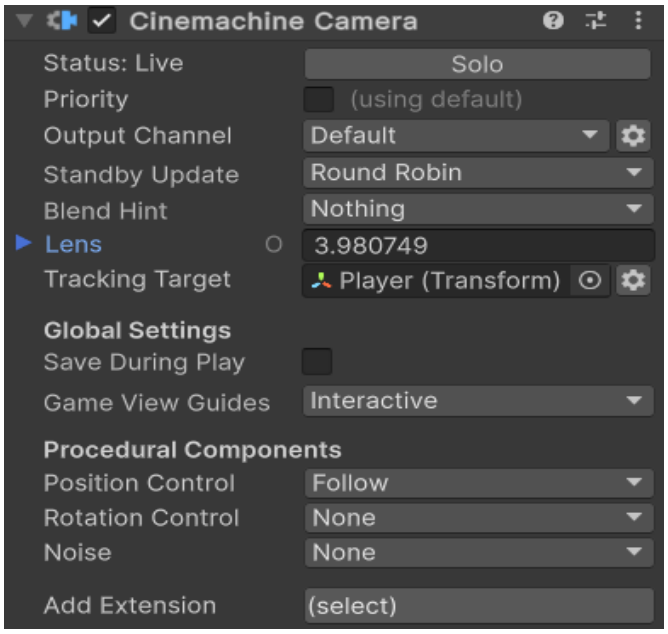
Một thành phần thiết yếu để hiển thị thế giới game cho người chơi. Camera giống như con mắt của người chơi, cho phép họ nhìn thấy mọi thứ trong khung cảnh của bạn:

- Render Type: Xác định cách camera hiển thị nội dung.
  + Base: Đây là loại camera cơ bản, nó hiển thị tất cả mọi thứ trong khung cảnh của nó. Bạn có thể thêm các camera khác để hiển thị nội dung bổ sung lên trên camera này.
  + Overlay: Camera này được sử dụng để hiển thị các nội dung "phủ lên trên" một camera Base khác, thường dùng để hiển thị giao diện người dùng (UI), bản đồ nhỏ hoặc các hiệu ứng đặc biệt mà không làm thay đổi cách hiển thị của camera chính.
  + Stack: Cho phép bạn tạo một "ngăn xếp" (stack) các camera. Bạn có thể sử dụng tính năng này để điều khiển

nhiều camera cùng một lúc, ví dụ như hiển thị một camera UI phủ lên trên một camera chơi game chính.

- Projection (Chiếu): Đây là phần bạn xác định cách thế giới 3D được chiếu lên màn hình 2D.
    + Perspective: Tạo ra một hiệu ứng 3D chân thực, nơi các vật thể ở xa trông nhỏ hơn. Thích hợp cho game 3D.
    + Orthographic: Các vật thể giữ nguyên kích thước bất kể khoảng cách. Thường được sử dụng cho game 2D hoặc các game 3D với góc nhìn cố định, ví dụ như các game chiến thuật từ trên xuống.
- Rendering (Kết xuất): Nhóm các tùy chọn nâng cao để điều chỉnh cách camera hiển thị hình ảnh. Bạn có thể điều chỉnh các thuộc tính như Clipping Planes (để xác định khoảng cách gần và xa nhất mà camera có thể nhìn thấy) và Culling Mask (để chọn những layer nào mà camera sẽ hiển thị).
- Stack (Ngăn xếp): Chỉ xuất hiện khi Render Type là Base. Bạn có thể thêm các camera Overlay vào đây để chúng được kết xuất lên trên camera chính.
- Environment (Môi trường): Điều chỉnh cách camera xử lý không gian trống không có vật thể.
- Output (Đầu ra): Nhóm các tùy chọn liên quan đến đầu ra của camera.

2.3.13 Cinemachine Camera



Hình 2.18: Hình ảnh về Cinemachine Camera

Một thành phần mạnh mẽ từ asset Cinemachine của Unity, được thiết kế để tự động hóa và tạo các chuyển động camera phức tạp mà không cần viết code:

- Các thuộc tính chính:
  + Status: Live: Cho biết trạng thái hiện tại của camera Cinemachine này.
    - Live: Camera này đang hoạt động và kiểm soát camera chính (Main Camera) của Unity.
    - Standby: Camera này đang ở trạng thái chờ và có thể được chuyển sang trạng thái Live bất kỳ lúc nào.
  + Priority (Ưu tiên): Khi bạn có nhiều camera Cinemachine trong một scene, Unity sẽ chọn camera có giá trị ưu tiên cao nhất để làm Live. Điều này cho phép bạn tạo ra các chuyển cảnh mượt mà giữa các camera khác nhau.
  + Output Channel: Xác định camera chính nào sẽ được điều khiển. Mặc định là Default, nghĩa là nó điều khiển camera chính duy nhất trong scene.
  + Standby Update: Điều chỉnh cách các camera ở chế độ chờ (standby) cập nhật.

- Round Robin: Cập nhật theo thứ tự vòng tròn để phân tán chi phí tính toán.
- Fixed Update: Cập nhật cùng với các bước vật lý.
- Never: Không bao giờ cập nhậ

+ Blend Hint: Gợi ý cho Cinemachine về cách tạo ra hiệu ứng chuyển cảnh (blend) khi chuyển từ camera này sang camera khác.
+ Lens: Nhóm các thuộc tính ống kính, tương tự như camera vật lý, để điều chỉnh góc nhìn và hiệu ứng. Giá trị bạn thấy, 3.980749, có thể là Orthographic Size (đối với 2D) hoặc Field of View (đối với 3D), xác định mức độ zoom của camera.
+ Tracking Target: Đây là thuộc tính quan trọng nhất. Bạn gán một GameObject (chẳng hạn như nhân vật người chơi) vào đây. Cinemachine sẽ tự động di chuyển camera để đi theo đối tượng đó.
+ Global Settings:
    - Save During Play: Khi bạn bật tùy chọn này, các thay đổi bạn thực hiện đối với camera trong chế độ Play sẽ được lưu lại sau khi bạn thoát khỏi chế độ đó.
    - Game View Guides: Tùy chọn này hiển thị các đường chỉ dẫn trong cửa sổ Game View để giúp bạn hình dung và căn chỉnh khung hình của camera.

- Các Thành Phần Theo Thủ Tục (Procedural Components):
    + Position Control: Xác định cách camera kiểm soát vị trí.
        - Follow: Camera sẽ đi theo đối tượng Tracking Target.
        - Orbital Transposer: Camera di chuyển xung quanh đối tượng mục tiêu.
        - Framing Transposer: Camera tự động điều chỉnh khoảng cách và vị trí để giữ cho đối tượng nằm trong khung hình.

+ Rotation Control: Xác định cách camera kiểm soát góc xoay.
    - Look At: Camera luôn xoay để hướng về phía đối tượng Tracking Target.
    - Do Nothing: Không điều chỉnh góc xoay.
+ Noise: Tạo hiệu ứng rung lắc camera, thường được dùng để mô phỏng sự chấn động, ví dụ như rung lắc của máy ảnh khi chạy hoặc khi xảy ra một vụ nổ.
+ Add Extension: Cho phép bạn thêm các chức năng mở rộng cho camera, ví dụ như tạo hiệu ứng chuyển động chậm (slow-motion) hoặc các hiệu ứng khác.

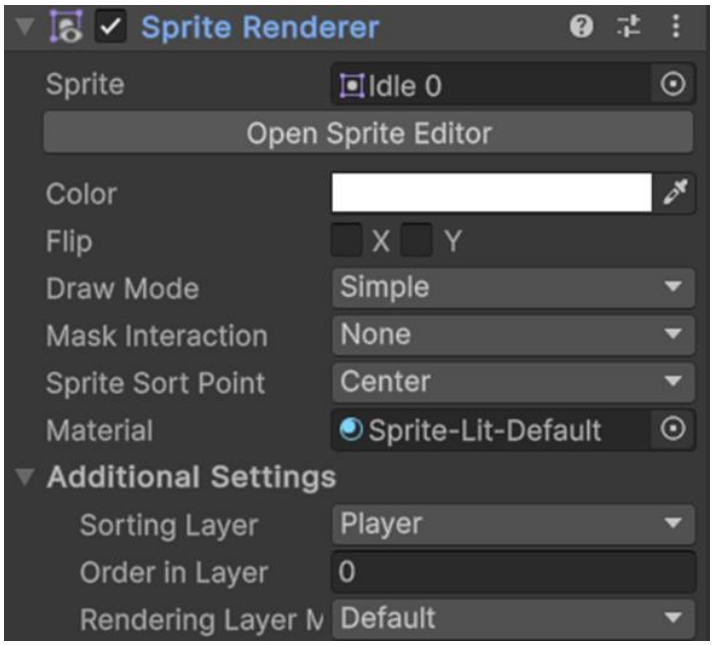
2.3.14 Transform

Hình 2.19: Hình ảnh về Transform

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

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

2.3.15 Sprite Renderer



Hình 2.20: Hình ảnh về Sprite Renderer

Sprite Renderer là một component quan trọng trong Unity được sử dụng để hiển thị các hình ảnh 2D (sprites) trong thế giới game.

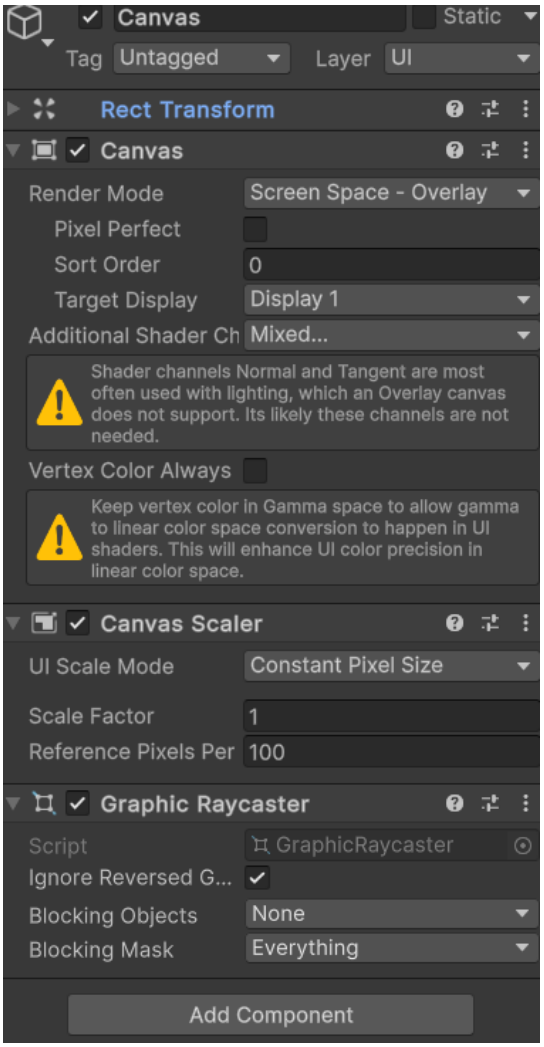
- Các thuộc tính chính:
  - + Sprite: Đây là nơi bạn gán hình ảnh (sprite) mà bạn muốn hiển thị trên màn hình. Bạn có thể chọn một sprite từ cửa sổ Project và kéo vào đây, hoặc sử dụng công cụ chọn để tìm kiếm.
  - + Open Sprite Editor: Nút này sẽ mở Sprite Editor, cho phép bạn cắt một hình ảnh lớn thành nhiều sprite nhỏ hơn. Ví dụ, bạn có thể cắt một tấm sprite chứa nhiều khung hình hoạt ảnh thành các sprite riêng lẻ.
  - + Color: Cho phép bạn thay đổi màu của sprite. Bạn có thể dùng nó để tạo hiệu ứng ánh sáng, làm mờ, hoặc thay đổi màu sắc của nhân vật/vật thể.
  - + Flip: Lật sprite theo chiều ngang hoặc chiều dọc.
    - X: Lật sprite theo trục X (ngang).
    - Y: Lật sprite theo trục Y (dọc).
  - + Draw Mode: Xác định cách sprite được hiển thị khi bạn thay đổi kích thước của nó.

- Simple: Chế độ mặc định, chỉ đơn giản là kéo giãn sprite.
- Sliced: Thường được dùng cho các phần tử UI hoặc các vật thể lặp lại. Nó chia sprite thành các ô và chỉ kéo giãn phần giữa để tránh làm biến dạng các góc.
- Tiled: Lặp lại sprite để lấp đầy một khu vực.

+ Mask Interaction: Xác định cách sprite tương tác với các Sprite Mask.
  - None: Không bị ảnh hưởng bởi Sprite Mask.
  - Visible Inside Mask: Chỉ hiển thị phần sprite nằm bên trong một Sprite Mask.
  - Visible Outside Mask: Chỉ hiển thị phần sprite nằm bên ngoài một Sprite Mask.
+ Sprite Sort Point: Dùng để xác định điểm trung tâm để sắp xếp các sprite khi chúng có cùng thứ tự.
  - Center: Sắp xếp dựa trên tâm của sprite.
  - Pivot: Sắp xếp dựa trên điểm Pivot (điểm neo) của sprite.
+ Material: Gán một vật liệu (material) cho sprite. Vật liệu này có thể chứa shader để tạo ra các hiệu ứng hình ảnh đặc biệt như ánh sáng, bóng đổ, hoặc các hiệu ứng đồ họa khác.

- Các thiết lập bổ sung:
  + Sorting Layer: Cho phép bạn gán sprite vào một lớp sắp xếp cụ thể. Các lớp này giúp bạn kiểm soát thứ tự hiển thị của các sprite. Ví dụ, bạn có thể đặt nhân vật vào một lớp "Player" và nền vào một lớp "Background" để đảm bảo nhân vật luôn hiển thị trên nền.
  + Order in Layer: Xác định thứ tự hiển thị của các sprite trong cùng một Sorting Layer. Giá trị cao hơn sẽ hiển thị trên các giá trị thấp hơn.

2.3.16 Canvas



Hình 2.21: Hình ảnh về Canvas

Canvas là một thành phần cốt lõi để xây dựng giao diện người dùng (UI). Canvas đóng vai trò là "bảng vẽ" cho tất cả các phần tử UI như nút, văn bản và hình ảnh.

Khi bạn tạo một Canvas mới trong Unity, nó thường đi kèm với ba component chính: Canvas, Canvas Scaler, và Graphic Raycaster:

- Rect Transform: Đây là component đầu tiên và quan trọng nhất cho bất kỳ phần tử UI nào. Khác với Transform thông thường (dành cho các đối tượng 3D), Rect Transform quản lý vị trí, kích thước và cách bố trí của các phần tử UI. Nó sử dụng các thuộc tính như Anchors (neo) và Pivot (điểm xoay) để đảm bảo UI của bạn hiển thị đúng trên nhiều độ phân giải màn hình khác nhau.
- Canvas: Component chính quản lý việc hiển thị UI.

+ Render Mode: Xác định cách Canvas được hiển thị so với các đối tượng trong game.
    - Screen Space - Overlay: Canvas được vẽ lên trên tất cả mọi thứ trong game. Đây là chế độ lý tưởng cho các HUD (giao diện hiển thị trên màn hình) và menu game.
    - Screen Space - Camera: Canvas được vẽ trên một camera cụ thể. UI sẽ hoạt động như một vật thể 3D và có thể bị các vật thể khác che khuất.
    - World Space: Canvas tồn tại như một vật thể 3D trong thế giới game. Rất hữu ích để tạo các bảng tên, thanh máu trên đầu nhân vật, hoặc bảng điều khiển trên màn hình game.
+ Pixel Perfect: Khi bật, Unity sẽ cố gắng căn chỉnh các phần tử UI với các pixel trên màn hình để hình ảnh sắc nét hơn, đặc biệt hữu ích với pixel art.
+ Sort Order: Dùng để xác định thứ tự vẽ khi có nhiều Canvas trong game. Canvas có Sort Order cao hơn sẽ được vẽ lên trên.

- Canvas Scaler: Component này giúp UI của bạn co giãn tự động để phù hợp với các độ phân giải màn hình khác nhau.
    + UI Scale Mode: Nó xác định cách giao diện người dùng (UI) của bạn điều chỉnh kích thước để hiển thị đúng và đẹp trên các màn hình có độ phân giải và tỷ lệ khung hình khác nhau.
        - Constant Pixel Size: Kích thước của các phần tử UI sẽ không thay đổi theo độ phân giải màn hình.
        - Scale With Screen Size: Chế độ phổ biến nhất. UI sẽ tự động co giãn để phù hợp với kích thước màn hình.
        - Constant Physical Size: UI giữ nguyên kích thước vật lý trên mọi thiết bị.
    + Reference Resolution: Đây là độ phân giải tham chiếu mà Unity sử dụng để tính toán tỷ lệ co giãn.

- Graphic Raycaster: Component này cho phép các phần tử UI tương tác với các sự kiện đầu vào như click chuột, chạm, hoặc rê chuột. Khi bạn tương tác với UI, Graphic Raycaster sẽ "bắn" một tia từ vị trí con trỏ để xác định phần tử nào đang được chọn. Nó giúp các nút bấm và thanh trượt hoạt động.

2.4. Tổng quan về Visual Studio

2.4.1 Giới thiệu

Hình 2.22: Logo Visual Studio

Visual studio là một phần mềm hỗ trợ đắc lực hỗ trợ công việc lập trình website. Công cụ này được tạo lên và thuộc quyền sở hữu của ông lớn công nghệ Microsoft. Năm 1997, phần mềm lập trình nay có tên mã Project Boston. Nhưng sau đó, Microsoft đã kết hợp các công cụ phát triển, đóng gói thành sản phẩm duy nhất..

Visual Studio là hệ thống tập hợp tất cả những gì liên quan tới phát triển ứng dụng, bao gồm trình chỉnh sửa mã, trình thiết kế, gỡ lỗi. Tức là, bạn có thể viết code, sửa lỗi, chỉnh sửa thiết kế ứng dụng dễ dàng chỉ với 1 phần mềm Visual Studio mà thôi. Không dừng lại ở đó, người dùng còn có thể thiết kế giao diện, trải nghiệm trong Visual Studio như khi phát triển ứng dụng Xamarin, UWP bằng XAML hay Blend vậy.

2.4.2 Các tính năng của Visual Studio

Tính đến nay, Visual Studio vẫn được coi là phần mềm lập trình hệ thống hàng đầu, chưa có phần mềm nào có thể thay thế được nó. Được đánh giá cao như vậy bởi Visual Studio sở hữu nhiều tính năng cực kỳ hấp dẫn. Cụ thể:

- Đa nền tảng: Phần mềm lập trình Visual Studio của Microsoft hỗ trợ sử dụng trên nhiều nền tảng khác nhau. Không giống như các trình

viết code khác, Visual Studio sử dụng được trên cả Windows, Linux và Mac Systems. Điều này cực kỳ tiện lợi cho lập trình viên trong quá trình ứng dụng.

- Đa ngôn ngữ lập trình: Không chỉ hỗ trợ đa nền tảng, Visual Studio cũng cho phép sử dụng nhiều ngôn ngữ lập trình khác nhau từ C#, F#, C/C++, HTML, CSS, Visual Basic, JavaScript,… Bởi vậy, Visual Studio có thể dễ dàng phát hiện và thông báo cho bạn khi các chương trình có lỗi.
- Hỗ trợ website: Visual Studio code cũng hỗ trợ website, đặc biệt trong công việc soạn thảo và thiết kế web.
- Kho tiện ích mở rộng phong phú: Mặc dù Visual Studio có hệ thống các ngôn ngữ hỗ trợ lập trình khá đa dạng. Nhưng nếu lập trình viên muốn sử dụng một ngôn ngữ khác, bạn có thể dễ dàng tải xuống các tiện ích mở rộng. Tính năng hấp dẫn này được hoạt động như một phần chương trình độc lập nên không lo làm giảm hiệu năng của phần mềm.
- Lưu trữ phân cấp: Phần lớn các tệp dữ liệu đoạn mã của Visual Studio đều được đặt trong các thư mục tương tự nhau. Đồng thời, Visual Studio cũng cung cấp một số thư một cho các tệp đặc biệt để bạn lưu trữ an toàn, dễ tìm, dễ sử dụng hơn.
- Kho lưu trữ an toàn: Với Visual Studio, bạn có thể hoàn toàn yên tâm về tính lưu trữ, bởi phần mềm đã được kết nối GIT và một số kho lưu trữ an toàn được sử dụng phổ biến hiện nay.
- Màn hình đa nhiệm: Visual Studio sở hữu tính năng màn hình đa nhiệm, cho phép người dùng mở cùng lúc nhiều tập tin, thư mục dù chúng có thể không liên quan tới nhau
- Hỗ trợ viết code: Khi sử dụng code vào trong lập trình, với Visual Studio, công cụ này có thể đề xuất tới các lập trình viên một số tùy chọn thay thế nhằm điều chỉnh đôi chút để đoạn code áp dụng thuận tiện hơn cho người dùng.
- Hỗ trợ thiết bị đầu cuối: Phần mềm Visual Studio cũng tích hợp các loại thiết bị đầu cuối, giúp người dùng không cần chuyển đổi giữa hai màn hình hay trở về thư mục gốc khi thực hiện một thao tác cần thiết nào đó.

- Hỗ trợ Git: Do kết nối với GitHub nên Visual Studio cho phép hỗ trợ sao chép, kéo thả trực tiếp. Các mã code này sau đó cũng có thể thay đổi và lưu lại trên phần mềm.
- Intellisense: Tính năng nhắc Intellisense được sử dụng hầu hết trong các phần mềm lập trình, bao gồm cả Visual Studio. Tuy nhiên, so với các trình viết mã, Visual Studio vẫn được đánh giá cao về tính chuyên nghiệp. Đặc biệt, tính năng này còn có thể phát hiện tất cả các đoạn mã không đầy đủ, nhắc lập trình viên, gợi ý sửa đổi, khai báo biến tự động trong trường hợp lập trình viên quên, giúp bổ sung cú pháp còn thiếu,…
- Tính năng comment: Một tính năng cũng khá hay ho, hỗ trợ cho người lập trình trong trường hợp "nhớ nhớ quên quên" đó là tính năng bình luận. Tính năng này cho phép lập trình viên để lại nhận xét, giúp dễ dàng ghi nhớ công việc cần hoàn thành, không bỏ sót công đoạn nào.

CHƯƠNG 3: XÂY DỰNG GAME 2D FOR GRANDPA

3.1. Giới thiệu tổng quan

3.1.1 Thông tin game

Hình 3.1: Logo game For Grandpa

For GrandPa là một trò chơi 2D thuộc thể loại hành động, phiêu lưu. Đồ họa được thiết kế đơn giản với đồ họa pixel, tạo cảm giác cổ điển nhưng thân thuộc cho người chơi. Hiện tại được phát triển cho nền tảng thiết bị PC hệ điều hành Window.

3.1.2 Thể loại game và yếu tố game

❖ Đây là một tựa game 2D thuộc thể loại hành động, phiêu lưu.

- Khái niệm game hành động:

  Game hành động (Action games) là tựa game mà trọng tâm chính là phản xạ, kỹ năng và tốc độ của người chơi. Trong các trò chơi này, người chơi thường phải thực hiện các hành động liên tục như chiến đấu, né tránh, bắn súng, đánh nhau hoặc chạy nhảy để vượt qua các thử thách và kẻ địch.

  Gameplay của game hành động thường yêu cầu sự nhanh nhạy, chính xác và phản xạ tốt. Môi trường trong game có thể thay đổi liên tục với kẻ địch, chướng ngại vật và các boss, tạo cảm giác căng thẳng và hấp dẫn. Ngoài ra, thể loại này còn thường kết hợp với các yếu tố của game phiêu lưu, nhập vai hay bắn súng, nhằm tăng chiều sâu trải nghiệm và sự đa dạng trong lối chơi.

- Khái niệm game phiêu lưu:

Trò chơi phiêu lưu hay trò chơi mạo hiểm là một thể loại video game mà trong đó giả định người chơi là nhân vật chính trong một câu chuyện có tính tương tác tiến triển theo hướng khám phá và vượt qua thử thách. Game thuộc thể loại này tập trung vào xây dựng cốt truyện, cho phép nó được diễn dịch sang một phạm vi rộng lớn loại hình truyền thông mang tính tường thuật như sách vở hay điện ảnh, và bao hàm hàng loạt thể loại văn chương. Gần như toàn bộ trò chơi phiêu lưu (dưới hình thức văn bản hay đồ họa) được thiết kế cho một người chơi, bởi vì chúng nhấn mạnh vào kịch bản và nhân vật, khiến cho nhiều người chơi sẽ trở nên khó khăn.

❖ Game mang lại nhiều yếu tố như:

- Tăng khả năng quan sát và sự nhanh nhẹn.
- Cốt truyện mang tính nhân văn.
- Mang tính giải trí, có thể chơi lúc rảnh rỗi.
- Độ khó vừa phải, tính thử thách cao.

3.1.3 Đối tượng chơi

Trò chơi được phát triển cho những độ tuổi trẻ để giải trí, thư giãn sau những phút học tập, làm việc mệt mỏi. Với lối chơi game đơn giản bằng các thao tác nhấn nút di chuyển theo hướng mũi tên. Độ tuổi tối thiểu để có thể thao tác game phù hợp là từ trên 6 tuổi.

3.1.4 Nền tảng

Game được xây dựng bằng Unity nên có khả năng phát triển tốt trên các nền tảng lớn. Nhưng hiện tại game tập trung chủ yếu vào nền nảng Window(Laptop và PC).

3.2 Kịch bản game

3.2.1 Mô tả

Nhân vật chính của game là một chú thỏ mang tên Kitty sống cùng người ông thỏ già nua nhưng tràn đầy yêu thương. Một buổi sáng nọ, bỗng dưng ông biến mất, chú thỏ nghe mọi người nói ông của cậu bị cuốn lên những hòn đảo lơ lửng ở trên trời. Và với sự dũng cảm và quyết tâm của mình cậu đã bước lên trên con đường đi tìm ông của mình.

Người chơi điều khiển nhân vật chính vượt qua các chướng ngại vật, thu thập coin và đi tìm ông theo những hòn đảo trên trời, sẽ có những kẻ địch, chướng ngoại vật và địa hình mới lạ tạo sự hứng thú và tò mò cho người chơi.

3.2.2 Luật chơi

Game cho phép người chơi tự do di chuyển theo không gian game quy ước. Không có giới hạn thời gian trong một màn chơi.

Người chơi có thể nhảy 2 lần, tận dụng địa hình để vượt qua những chướng ngại vật và tìm thấy ông mình ở đích đến.

Điều kiện thắng:

Người chơi thu thập các coin trên đường đi, tránh xa kẻ địch và chướng ngại vật để tìm thấy ông mình ở đích đến và chiến thắng và có thể tiếp tục màn chơi tiếp theo.

Điều kiện thua:

Nếu người chơi vi phạm điều kiện dưới đây sẽ được tính là thua.

➢ Số lượng máu của người chơi bằng 0.

➢ Người chơi rơi khỏi map.

Mỗi màn chơi bị thua cuộc phải chơi lại mới sang màn tiếp theo.

3.2.3 Tính điểm

Chú thỏ Kitty cung cấp điểm số cho người chơi để tăng cảm giác đạt được thành tựu qua mỗi màn thông qua số coin đã nhặt.

3.2.4 Tương tác và điều khiển game

Tất cả các thao tác điều khiển trò chơi đều được thực hiện bởi bàn phím (PC). Người chơi sử dụng bàn phím máy tính di chuyển.

- Di chuyển sang trái: Phím mũi tên trái hoặc phím A.
- Di chuyển sang phải: Phím mũi tên phải hoặc phím D.
- Nhảy: Phím SPACE, phím mũi tên lên.

3.2.5 Các phần tử của game (Game Elements)

3.2.5.1 Player

Player: nhân vật chính là chú thỏ tên Kitty mà người chơi cần điều khiển.

- Ban đầu có số lượng máu là 100. Bị trừ máu khi chạm bẫy hoặc kẻ địch một lượng máu là 25.

Hình 3.2: Player

3.2.5.2 Enemy

Enemy: Kẻ địch gây nguy hiểm cho người chơi. Gồm 2 loại:

- **Enemy1 (Kẻ địch trên mặt đất):**

Hình 3.3: Enemy1

- Enemy1:.Khiến người chơi mất máu khi tiếp xúc mức sát thương là 25 máu.
- Khi tiếp xúc với người chơi sẽ không bị đi sai quỹ đạo và luôn đi theo quỹ đạo đã được chỉnh sẵn trên mặt đất.

- **Fly Enemy (Kẻ địch bay trên trời):**

Hình 3.4: Fly Enemy

- Fly Enemy: Khiến người chơi mất máu khi tiếp xúc mức sát thương là 25 máu.
- Fly Enemy khi tiếp xúc với người chơi sẽ không bị đi sai quỹ đạo và luôn đi theo quỹ đạo đã được chỉnh sẵn trên ở trên trời.

3.2.5.3 Moving Platform

Moving Platform: Địa hình hỗ trợ người chơi di chuyển qua khu vực nguy hiểm hoặc địa hình cao. Gồm 2 loại:

- **MovingBlock**
- Di chuyển đoạn xa theo quỹ đạo ngang. Người chơi có thể đứng trên đối tượng này để đi qua những khu vực nguy hiểm.

Hình 3.5: MovingBlock

- **MovingBlockUD**
- Di chuyển đoạn xa theo quỹ đạo thẳng đứng. Người chơi có thể đứng trên đối tượng này để đi qua những khu vực cao hoặc thấp.

Hình 3.6: MovingBlockUD

3.2.5.4 Trap

Trap: đối tượng gây nguy hiểm tới người chơi. Gồm 2 loại:

- **Spike (Gai nhọn)**

Hình 3.7: Spike

➢ Được chỉnh ở vị trí cố định. Khi người chơi chạm vào sẽ bị trừ 25 máu.

- **Super Spike (Gai nhọn khổng lồ)**

Hình 3.8: Super Spike

➢ Được chỉnh ở vị trí cố định, có kích thước to gấp 5 lần Spike. Khi người chơi chạm vào sẽ bị trừ 25 máu.

3.2.5.5 BouncePad

BouncePad: là GameObject khi Player tiếp xúc sẽ đẩy Player lên một độ cao bằng 3 lần nhảy bình thường.

Hình 3.9: Bounce Pad

3.2.5.6 Items

Items: là GameObject khi Player tiếp xúc sẽ biến mất và thêm cho Player một chỉ số dựa theo loại GameObject trong Items.

Coin

Hình 3.10: Coin

- Xuất hiện ngẫu nhiên trên đường đi, thu thập chúng để cuối màn đánh giá thành tích của người chơi.

3.2.5.7 Finish

Finish: là điểm đến mà Player hướng tới khi Player tiếp xúc hướng Player đến màn hình chiến thắng.

GrandPa

Hình 3.11: GrandPa

- Xuất hiện ở cuối đường đi, tiếp xúc để chuyển sang màn hình chiến thắng, có thể chọn màn tiếp theo hoặc quay về menu chính.

3.2.5.8 Level Board

Level Board: là GameObject ghi lại Level hiện tại bạn đang chơi.

Hình 3.12: LevelBoard

3.2.6 Thiết kế các level của game

For GrandPa hiện tại có 5 màn chơi, mỗi màn sẽ xuất hiện những kẻ địch và địa hình mới để tăng tính thử thách cho người chơi. Ngoài ra, trên đường đi sẽ bắt gặp một số cạm bẫy gây cản trở người chơi, cần di chuyển thông minh để vượt qua nó.

Từ các phần tử game được nêu ở phần 3.2.5 ta sắp xếp và thiết kế các level gồm có Player, vật phẩm Coin, Level Board, Finish còn các Enemy, Trap, Moving Platform, BouncePad có số lượng cụ thể như sau:

- Level 1: 1 Enemy, 1 Trap, 1 Moving Platform.

- Level 2: 2 Enemy, 1 Trap, 1 Moving Platform.
- Level 3: 1 Enemy, 3 Trap, 1 Bounce Pad.
- Level 4: 2 Enemy, 3 Trap, 1 Moving Platform, 2 Bounce Pad.
- Level 5: 3 Enemy, 6 Trap, 2 Moving Platform, 1 Bounce Pad

3.2.7 Các cơ chế của game (Game Mechanic)

Game cho phép người chơi tự do di chuyển theo không gian game quy ước. Không có giới hạn thời gian trong một màn chơi. Người chơi tự do trải nghiệm trò chơi theo cách mình muốn.

Người chơi chỉ có thể nhảy được 2 lần liên tục và khi chạm đất mới có thể nhảy tiếp vậy nên người chơi cần căn thời gian khi nào nên nhảy sao cho đạt được hiệu quả tốt nhất, người chơi cũng phải tận dụng địa hình, tránh các Enemy và Trap.

Khi tiếp xúc với Enemy hoặc Trap thì người chơi sẽ bị nhảy lên và mất máu nên cần lưu ý để không bị văng ra nơi không mong muốn.

Mỗi màn chơi có một số lượng vật phẩm coin nhất định. Người chơi cần thu thập chúng, gặp lại ông cuối màn để chiến thắng. Cuối màn chơi sẽ được tổng kết trình bày ở mục 3.2.3.

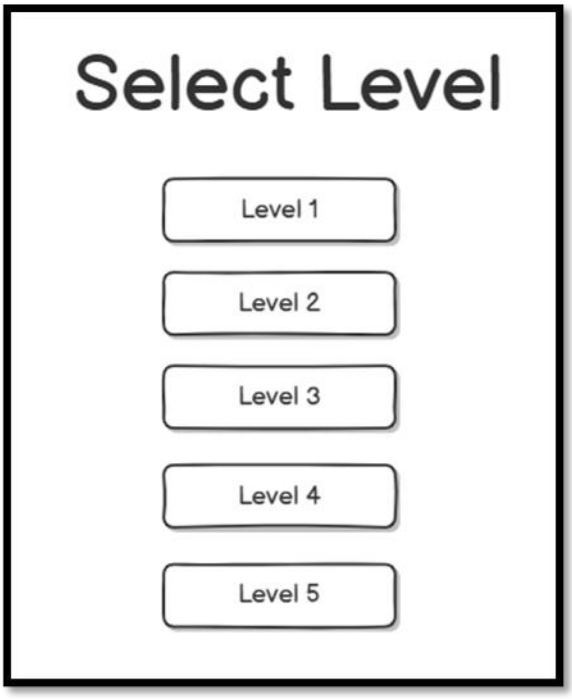
3.3 Thiết kế giao diện các màn hình

3.3.1 Màn hình Main Menu

Giao diện màn hình có 2 button: Start và Quit Game được hiển thị như bên dưới.

Hình 3.13: Giao diện Main Menu

Click button Start : Sẽ chuyển sang màn hình chọn Level hiển thị màn hình chọn màn chơi.



Hình 3.14: Giao diện Select Level

3.3.2 Màn hình game

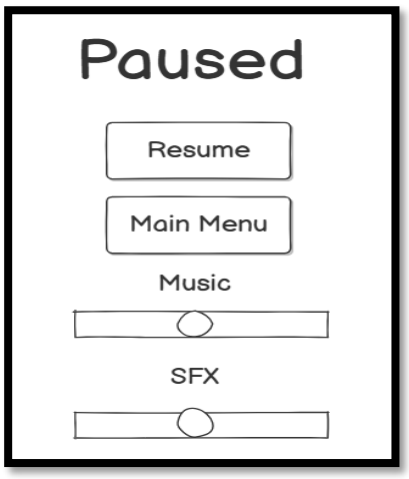
Sau khi chọn level ở Hình 3.14, màn hình game hiện thị như sau:

- Thông tin trạng thái máu của Player hiển thị góc trái trên màn hình.
- Số coin Player thu thập được hiện ở góc trái dưới màn hình.

Hình 3.15: Giao diện màn hình game

3.3.3 Màn hình Paused Menu

Sau khi nhấn Esc ở bàn phím, màn hình Paused Menu trong game hiển thị như sau:



Hình 3.16: Giao diện Paused Menu

- Click button Resume: để tiếp tục game và hiển thị màn hình game như Hình 3.15.
- Click button Main Menu: để thoát màn chơi và hiển thị màn hình Main Menu như Hình 3.13.
- Có thể điều chỉnh âm thanh nhạc nền thông qua thanh Music Slider và điều chỉnh SFX thông qua thanh SFX Slider. Kéo sang trái là giảm âm lượng, kéo sang phải là tăng âm lượng.

3.3.4 Màn hình chiến thắng

Sau khi đến đích, màn hình chiến thắng trong game hiển thị như sau:

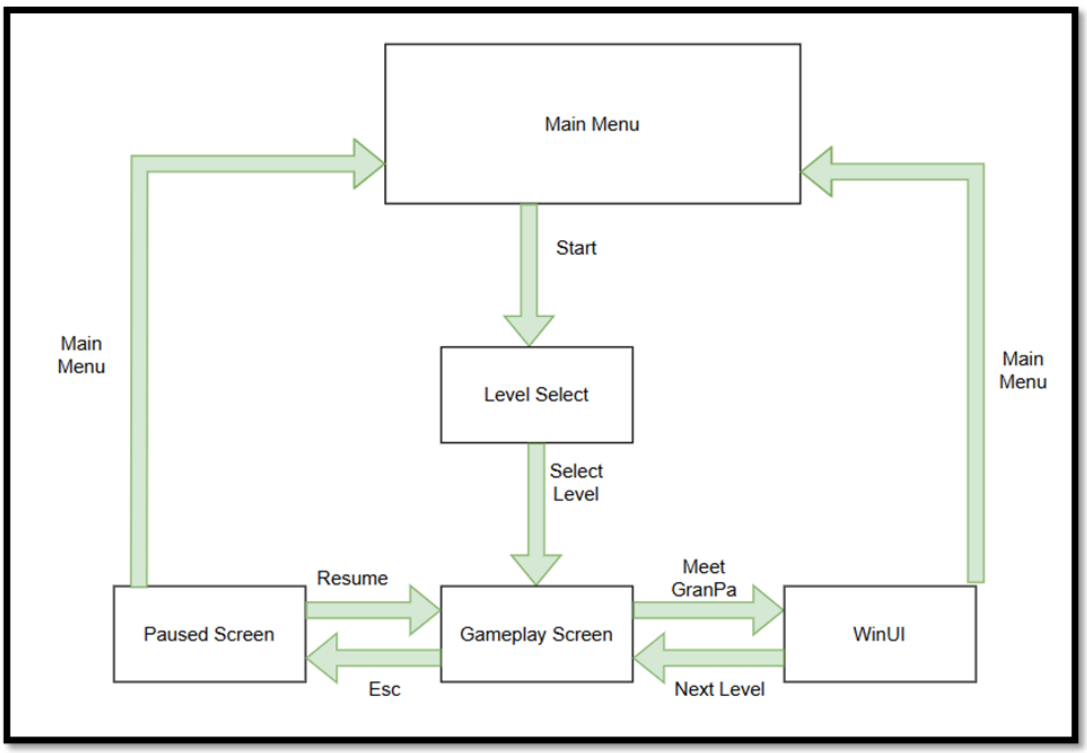
Hình 3.17: Giao diện màn hình chiến thắng

- Click button Next Level: hiển thị màn game như Hình 3.15.

- Click button Main Menu: để thoát màn chơi và hiển thị màn hình Main Menu như Hình 3.13.

3.3.5 Story board

Từ các giao diện màn hình trên ta tổng hợp được Storyboard như sau:



Hình 3.18: Story Board của game

3.4 Tài nguyên

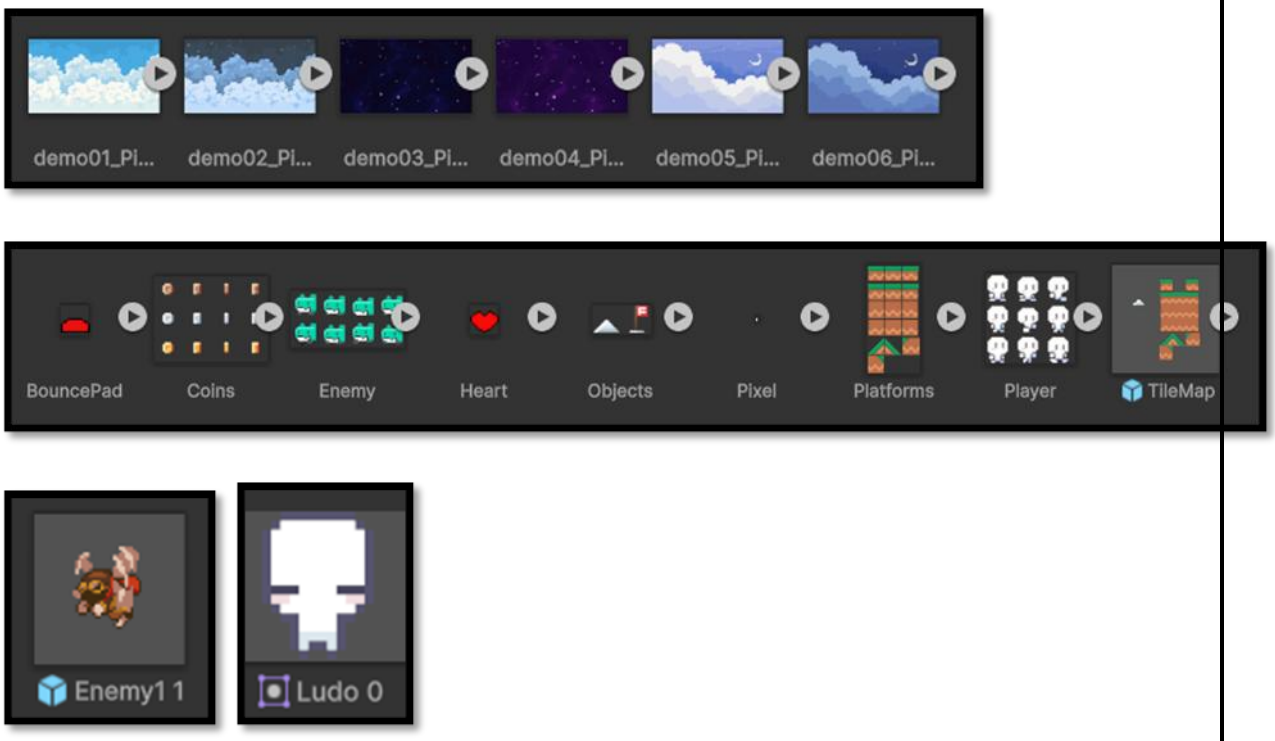
3.4.1 Hình ảnh

Một số hình ảnh về Enemy, Trap, Background, Coin, Player:
https://assetstore.unity.com/packages/2d/characters/simple-2d-platformer-assets-pack-188518

https://assetstore.unity.com/packages/2d/environments/pixel-skies-demo-background-pack-226622

https://assetstore.unity.com/packages/2d/characters/sunny-land-forest-108124

https://assetstore.unity.com/packages/2d/characters/top-down-2d-rpg-assets-pack-188718



Hình 3.19: Một số hình ảnh sử dụng trong game

3.4.2 Âm thanh

Sử dụng các âm thanh nhạc nền, nhảy, thu thập vật phẩm,… giúp game sinh động hơn

STT	Tên	Mô tả	Nguồn
1	CoinCollect	Âm thanh khi thu thập coin	https://assetstore.unity.com/packages/audio/sound-fx/true-8-bit-sound-effect-collection-lite-version-264063
2	BackGroundMusic	Âm thanh nhạc nền	https://assetstore.unity.com/packages/audio/sound-fx/true-8-bit-sound-effect-collection-lite-version-264063
3	DamageSound	Âm thanh khi bị trừ HP	https://assetstore.unity.com/packages/audio/sound-fx/true-8-bit-sound-

			effect-collection-lite-version-264063
4	WinSound	Âm thanh khi Player chạm vào Finish	https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116

Bảng 3.1: Bảng thông tin âm thanh game

3.5 Các kỹ thuật để xây dựng game

3.5.1 Các kỹ thuật với Player

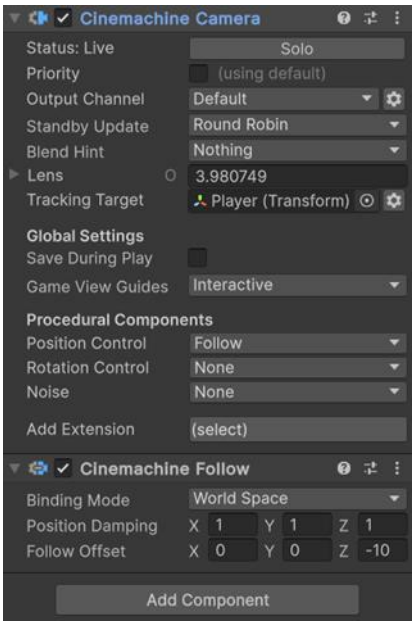
3.5.1.1 Di chuyển và double jumps

```
void Update()
{
    //move left and right
    float moveInput = Input.GetAxis("Horizontal");
    rb.linearVelocity = new Vector2(moveInput * moveSpeed, rb.linearVelocity.y);

    // flip player theo hướng di chuyển
    if (moveInput > 0.01f)
        spriteRenderer.flipX = false; // quay phải
    else if (moveInput < -0.01f)
        spriteRenderer.flipX = true;  // quay trái

    if (isGrounded)
    {
        extraJumps = extraJumpsValue;
    }

    //jump
    if (Input.GetKeyDown(KeyCode.Space) || Input.GetKeyDown(KeyCode.UpArrow))
    {
        if (isGrounded)
        {
            rb.linearVelocity = new Vector2(rb.linearVelocity.x, jumpForce);
            audioManager.PlaySFX(audioManager.jump);
        }
        else if (extraJumps > 0)
        {
            rb.linearVelocity = new Vector2(rb.linearVelocity.x, jumpForce);
            extraJumps--;
            audioManager.PlaySFX(audioManager.jump);
        }
    }

    SetAnimation(moveInput);

    healthImage.fillAmount = health / 100f;
}

Unity Message | 0 references
private void FixedUpdate()
{
    isGrounded = Physics2D.OverlapCircle(groundCheck.position, groundCheckRadius, groundLayer);
}
```

Hình 3.20: Player di chuyển và double jumps

- Player di chuyển trái phải và nhảy bằng bàn phím máy tính: Nhấn nút A, D, Space hoặc các phím mũi tên. Khi Player di chuyển theo hướng nào thì mặt Player sẽ nhìn theo hướng đó. Khi nhảy thì sẽ chạy SFX đã cài sẵn.
- Double jumps: nhấn mũi tên hướng lên hoặc phím Space 2 lần liên tục.

3.5.1.2 Camera theo dõi Player



Hình 3.21: Camera theo dõi Player

Kéo GameObject Player vào ô Tracking Target và Position Control chỉnh là "Follow"

3.5.1.3 Trạng thái khi tiếp xúc vật lý của Player

```
// get damage
Unity Message | 0 references
private void OnCollisionEnter2D(Collision2D collision)
{
    if (collision.gameObject.tag == "Damage")
    {
        audioManager.PlaySFX(audioManager.hurt);
        health -= 25;
        rb.linearVelocity = new Vector2(rb.linearVelocity.x, jumpForce);
        StartCoroutine(BlinkRed());

        if (health <= 0)
        {
            audioManager.PlaySFX(audioManager.death);
            Die();
        }
    }
    else if (collision.gameObject.tag == "BouncePad")
    {
        rb.linearVelocity = new Vector2(rb.linearVelocity.x, jumpForce * 2);
        audioManager.PlaySFX(audioManager.jump);
    }
    else if (collision.gameObject.CompareTag("FlyEnemy"))
    {
        // player nhảy lên khi chạm enemy
        rb.linearVelocity = new Vector2(rb.linearVelocity.x, jumpForce);
        StartCoroutine(BlinkRed());
        health -= 25;
    }
}
//red color when get damage
2 references
private IEnumerator BlinkRed()
{
    spriteRenderer.color = Color.red;
    yield return new WaitForSeconds(0.1f);
    spriteRenderer.color = Color.white;
}

1 reference
private void Die()
{
    UnityEngine.SceneManagement.SceneManager.LoadScene(UnityEngine.SceneManagement.SceneManager.GetActiveScene().name);
}
```

Hình 3.22: Trạng thái tiếp xúc vật lý của Player

- Player sau khi bị Enemy tấn công hoặc chạm bẫy thì sẽ mất 1 lượng máu là 25, nếu lượng máu <=0 thì Player sẽ chết và spawn Player ở vị trí khi bắt đầu vào Level.
- Player sau khi nhận sát thương sẽ nháy đỏ và nhảy lên, chạy SFX đã cài sẵn và trừ vào thanh máu 25 điểm HP.

3.5.2 Kỹ thuật xây dựng vật phẩm

Coin

```
private void Start()
{
    coinText = GameObject.FindWithTag("CoinText").GetComponent<TextMeshProUGUI>();
    audioManager = GameObject.FindGameObjectWithTag("Audio").GetComponent<AudioManager>();
}
Unity Message | 0 references
private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.gameObject.tag == "Player")
    {
        Player player = collision.gameObject.GetComponent<Player>();
        player.coins += 1;
        audioManager.PlaySFX(audioManager.getcoin);
        coinText.text = player.coins.ToString();
        Destroy(gameObject);
    }
}
```

Hình 3.23: Vật phẩm Coin

Khi collider của Player va chạm vào collider của Coin thì vật phẩm coin biến mất, màn hình game sẽ cộng thêm điểm coin. Và chạy SFX đã cài sẵn.

3.5.3 Các kỹ thuật với Enemy

Các hành vi của Enemy

```
void Start()
{
    spriteRenderer = GetComponent<SpriteRenderer>();
}


Unity Message | 0 references
void Update()
{
    if (Vector2.Distance(transform.position, points[i].position) < 0.25f)
    {
        i++;
        if (i == points.Length)
        {
            i = 0;
        }
    }

    transform.position = Vector2.MoveTowards(transform.position, points[i].position, speed * Time.deltaTime);

    //flip enemy
    spriteRenderer.flipX = (transform.position.x - points[i].position.x) < 0f;
}
```

Hình 3.24: Hành vi của Enemy1

- Enemy1:
    + Di chuyển đến điểm chỉ định: Enemy sẽ di chuyển đến 2 điểm đã chỉnh từ trước, khi đi hết điểm này thì sẽ chuyển sang điểm còn lại và liên tục lặp lại, Enemy1 sẽ di chuyển trên mặt đất.
    + Enemy sẽ được set Bodytype là Kinematic nhằm khi Player chạm vào thì Enemy1 sẽ không bị văng đi xa và đánh mất quỹ đạo.
    + Khi Enemy1 chuyển hướng di chuyển thì sẽ lật lại Enemy1 đó để hướng đi = trùng với hướng mắt của Enemy1.

```
void Start()
{
    spriteRenderer = GetComponent<SpriteRenderer>();

    rb = GetComponent<Rigidbody2D>();
    if (rb == null) rb = gameObject.AddComponent<Rigidbody2D>();

    // Kinematic → không bị physics tác động
    rb.bodyType = RigidbodyType2D.Kinematic;
    rb.gravityScale = 0f;
    rb.freezeRotation = true;

    if (points.Length == 0)
    {
        Debug.LogError("Chưa gán điểm bay!");
    }
}

Unity Message | 0 references
void Update()
{
    if (points.Length == 0) return;

    // target hiện tại
    Vector2 targetPos = new Vector2(points[i].position.x, transform.position.y); // chỉ di chuyển ngang

    // di chuyển ngang bằng MovePosition (không bị physics đẩy)
    rb.MovePosition(Vector2.MoveTowards(rb.position, targetPos, speed * Time.deltaTime));

    // khi tới target → chuyển sang point tiếp theo
    if (Mathf.Abs(rb.position.x - targetPos.x) < 0.05f)
    {
        i++;
        if (i >= points.Length) i = 0;
    }

    // flip sprite theo hướng di chuyển
    float dirX = targetPos.x - rb.position.x;
    spriteRenderer.flipX = dirX > 0;
}
```

Hình 3.25: Hành vi của Fly Enemy

- Fly Enemy:
    + Di chuyển đến điểm chỉ định: Enemy sẽ di chuyển đến 2 điểm đã chỉnh từ trước, khi đi hết điểm này thì sẽ chuyển sang điểm còn lại và liên tục lặp lại, chỉnh

Gravity Scale của Fly Enemy = 0 để không bị rơi xuống, Fly Enemy sẽ di chuyển ở trên trời.

+ Fly Enemy sẽ được set Bodytype là Kinematic nhằm khi Player chạm vào thì Fly Enemy sẽ không bị văng đi xa và đánh mất quỹ đạo.
+ Khi Fly Enemy chuyển hướng di chuyển thì sẽ lật lại Fly Enemy đó để hướng đi trùng với hướng mắt của Fly Enemy.

3.5.4 Kỹ thuật xây dựng bẫy

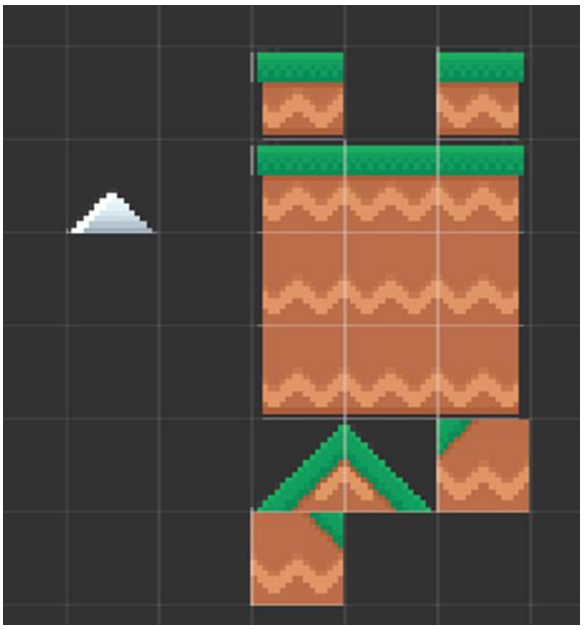
```
if (collision.gameObject.tag == "Damage")
{
    audioManager.PlaySFX(audioManager.hurt);
    health -= 25;
    rb.linearVelocity = new Vector2(rb.linearVelocity.x, jumpForce);
    StartCoroutine(BlinkRed());

    if (health <= 0)
    {
        audioManager.PlaySFX(audioManager.death);
        Die();
    }
}
```

Hình 3.26: Xây dựng bẫy

Khi collider của Player chạm collider có tag "Damage" thì Player nhận lượng sát thương damage một lượng 25 điểm HP. Và chạy SFX đã cài sẵn.

3.5.5 Kỹ thuật thiết kế map

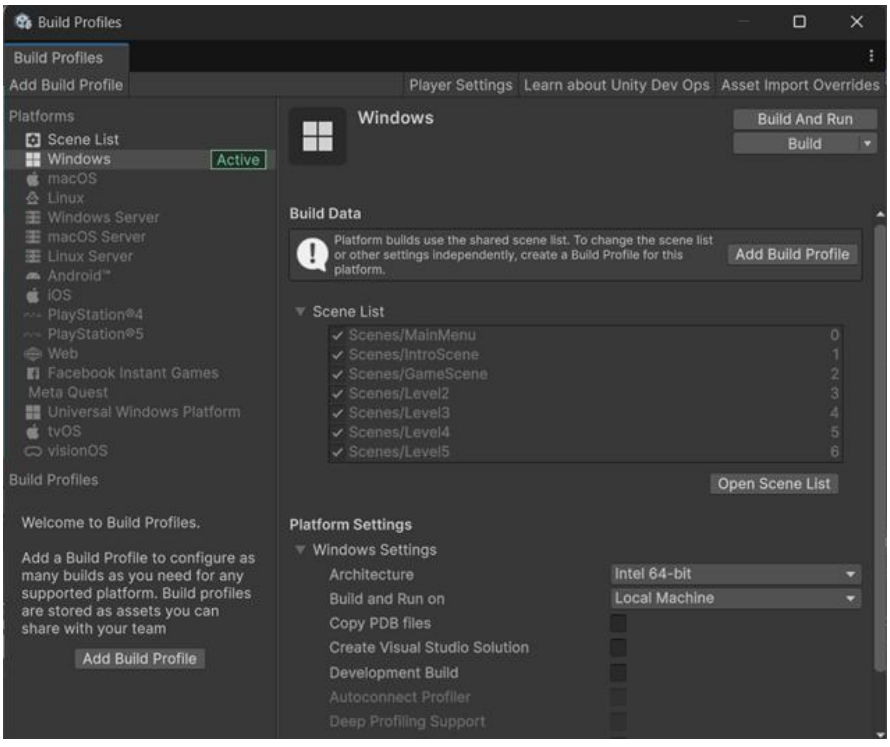
Trong phần mềm Unity: Chọn cửa sổ Window > 2D > Tile palette > Create New Palette > chọn vị trí lưu > Add các tilemap đã tải hoặc tự tạo vào Tile Palette đã tạo > Chọn tilemap phù hợp rồi vẽ lên map.



Hình 3.27: Thiết kế map

3.5.6 Xuất sản phẩm

Trong phần mềm Unity: Chọn cửa sổ File > Build setting > Tích chọn các Scenes in Build > ở Platform chọn PC nếu build bản cho máy tính, chọn Android nếu build bản cho điện thoại > Switch Platform > Build > Chọn vị trí lưu file game.



Hình 3.28: Xuất sản phẩm game

3.6 Kết quả thực nghiệm

Qua các kỹ thuật để xây dựng và thiết kế game trên, ta thu được các hình kết quả dưới đây:

3.6.1 Màn hình bắt đầu

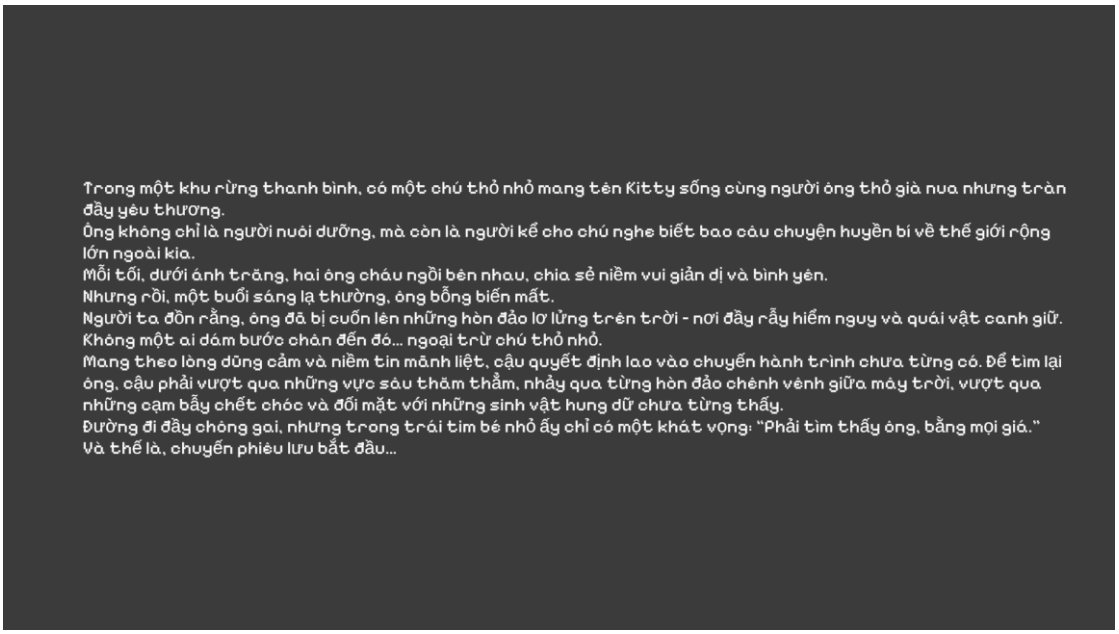
Qua các kỹ thuật để xây dựng và thiết kế game trên, ta thu được các hình kết quả dưới đây:

Hình 3.29: Màn hình Main Menu

Giao diện Select Level sau khi bấm nút Start.

Hình 3.30: Màn hình Select Level

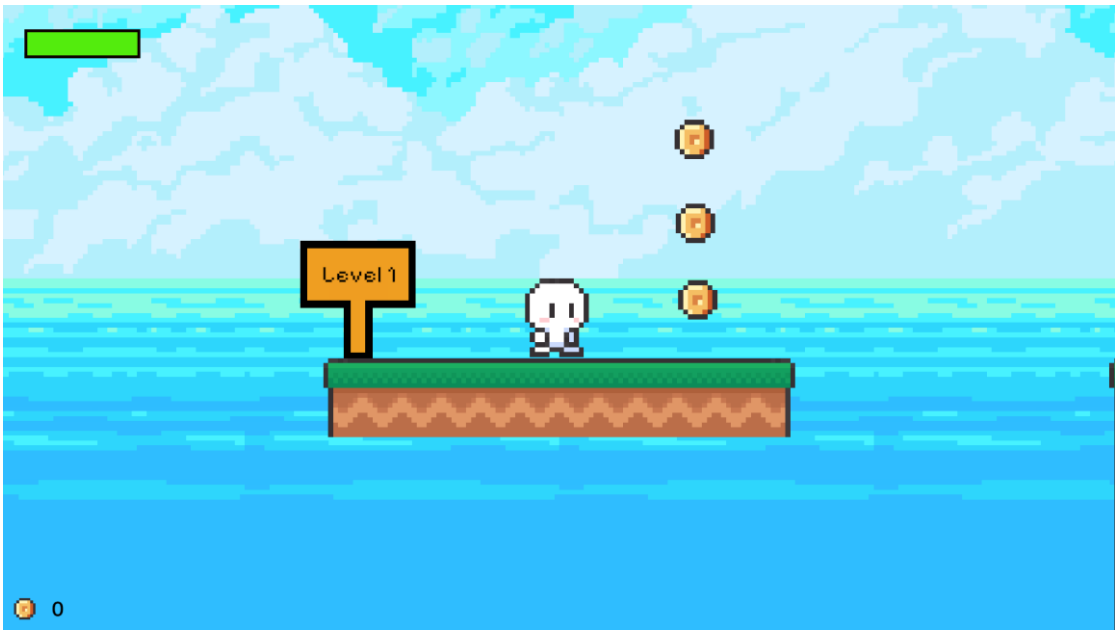
Giao diện Intro sau khi bấm vào Level 1, từ Level 2 trở đi thì không có Intro nữa.



Hình 3.31: Màn hình Intro

3.6.2 Màn hình chơi game

Giao diện game sau khi click qua màn hình Intro.



Hình 3.32: Màn hình gameplay

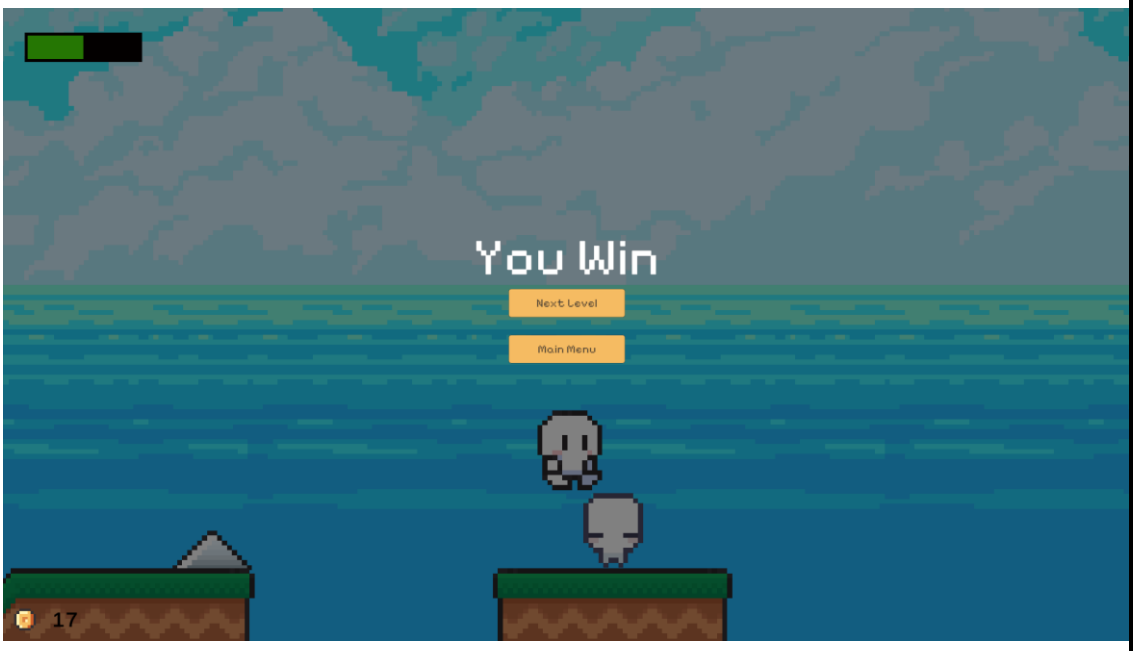
3.6.3 Màn hình Paused Menu

Sau khi bấm nút Esc ở bàn phím sẽ chuyển tới giao diện Paused Menu. Ở màn hình này ta có thể điều chỉnh âm lượng thông qua Music Slider và SFX Slider.

Hình 3.33: Màn hình Paused Menu

3.6.4 Màn hình thắng

Sau khi gặp ông nội của mình ở vạch đích thì sẽ chuyển sang màn hình chiến thắng.



Hình 3.34: Màn hình chiến thắng

CHƯƠNG 4: KIỂM THỬ GAME 2D FOR GRANDPA

4.1 Mục đích kiểm thử

Sau khi gặp ông nội của mình ở vạch đích thì sẽ chuyển sang màn hình chiến thắ Kiểm thử (testing) là bước quan trọng trong quá trình phát triển game, nhằm đảm bảo rằng sản phẩm hoạt động ổn định, đúng yêu cầu và mang lại trải nghiệm tốt cho người chơi. Quá trình kiểm thử giúp phát hiện lỗi (bug), điểm chưa hợp lý trong gameplay và đề xuất cải tiến trước khi phát hành.

4.2 Quy trình kiểm thử

Quy trình kiểm thử được thực hiện theo các bước:

- Chuẩn bị: Xác định mục tiêu kiểm thử, chọn người tham gia thử nghiệm.
- Thực hiện: Người kiểm thử tiến hành trải nghiệm game, ghi nhận lỗi và cảm nhận.
- Ghi nhận kết quả: Tập hợp dữ liệu từ các lỗi, sự cố và phản hồi.
- Đề xuất giải pháp: Phân tích nguyên nhân, đưa ra hướng khắc phục để cải thiện game.

4.3 Kết quả kiểm thử

Nội dung kiểm thử	Kết quả phát hiện	Giải pháp đề xuất
Nhân vật di chuyển trái/ phải.	Đôi khi bị khựng.	Tối ưu lại script điều khiển chuyển động.
Nhảy 2 lần liên tiếp.	Không thực hiện được double jump.	Kiểm tra lại input jump và biến trạng thái.
Hiển thị điểm số.	Điểm số chưa cập nhật khi ăn vật phẩm.	Gắn sự kiện tăng điểm vào Trigger của vật phẩm.

Bảng 4.1: Bảng kiểm thử của Nguyễn Duy Thái

Nội dung kiểm thử	Kết quả phát hiện	Giải pháp đề xuất
Âm thanh nền.	Bị lặp quá to gây khó chịu.	Thêm tùy chỉnh âm lượng trong menu.
Hiệu ứng va chạm.	Nhân vật va chạm với Enemy nhưng không thấy cảnh báo.	Xem lại hiệu ứng khi Player tương tác với collider của Enemy

Mức độ khó Level 2.	Quá khó so với Level 1.	Cân bằng độ khó, giảm số lượng kẻ địch.

Bảng 4.2: Bảng kiểm thử của Phan Văn Thức

Nội dung kiểm thử	Kết quả phát hiện	Giải pháp đề xuất
Hiển thị UI.	Thanh máu che mất nhân vật khi nhảy cao.	Di chuyển thanh máu lên góc màn hình.
Enemy và Trap	Thiếu Enemy và Trap	Thêm Enemy và Trap để game đa dạng hơn.
Nhảy vào Bounce Pad	Không thấy bật lên	Cần xem lại Collider của BouncePad.

Bảng 4.3: Bảng kiểm thử của Võ Trung Hiếu

4.4 Kết quả kiểm thử

Qua các kết quả kiểm thử từ nhiều người chơi, có thể thấy game đã hoạt động tương đối ổn định nhưng vẫn còn tồn tại một số lỗi về điều khiển, UI và cân bằng độ khó. Các giải pháp đã được đề xuất tập trung vào:

- Tối ưu lại code điều khiển nhân vật.
- Cải thiện giao diện người dùng và hiệu ứng hình ảnh.
- Cân bằng độ khó giữa các màn chơi.
- Bổ sung các tùy chọn về âm thanh, hiệu ứng để tăng trải nghiệm người chơi.

Việc áp dụng những giải pháp này sẽ giúp game trở nên mượt mà, hấp dẫn và thân thiện hơn với người chơi trước khi phát hành chính thức.

KẾT LUẬN

1. Kết quả đạt được

Từ đề tài đồ án game For GrandPa, em đã trình bày được cơ sở lý thuyết, công cụ để thực hiện và vận dụng kiến thức cơ sở học được ở trường giúp giải quyết đề tài này. Cải thiện việc tự học, tự nghiên cứu cách xây dựng, thiết kế game và lập trình game 2D trong Unity. Qua đó thành thạo việc tạo và sử dụng các hoạt ảnh animation, chỉnh sửa animator. Trong sản phẩm game thì: Player và Enemy có sự tương tác; giao diện menu bắt mắt, âm thanh game sống động, màn chơi phong phú; cạm bẫy đa dạng.

2. Chưa đạt được

Vì thời gian và khả năng có hạn, có một vài thứ chưa đạt được mong muốn như: hiệu ứng UI còn hạn chế; số lượng màn chơi không nhiều, mới chỉ có 5 màn; số lượng kẻ địch, cạm bẫy hơi ít.

3. Thuận lợi

Để xây dựng sản phẩm này, em rất may mắn khi có kiến thức cơ sở được học ở trường về thiết kế game trên Unity và được giáo viên chỉ bảo và bạn bè giúp đỡ nhiệt tình. Ngoài ra thì tài liệu trên internet rất nhiều, thoải mái nghiên cứu, tìm hiểu cách xây dựng, thiết kế game và lập trình game 2D trong thời gian làm đồ án.

4. Khó khăn

Bên cạnh đó, do khả năng tiếng Anh chưa tốt, nên việc đọc hiểu các tài liệu ngôn ngữ này còn chậm. Chưa có sẵn hình ảnh, hoạt ảnh đẹp và phù hợp để tạo animation và xây dựng các phần tử trong game. Phải mất nhiều thời gian chỉnh sửa ảnh chưa hoàn thiện kiếm được trên internet, căn chỉnh vị trí, kích thước phù hợp với game.

5. Kinh nghiệm rút ra

Biết cách sắp xếp thời gian hợp lý nếu không sẽ gặp rất nhiều khó khăn. Hiểu được việc tự học và nghiên cứu là vô cùng quan trọng. Nắm bắt được cách chỉnh sửa hoạt ảnh, nâng cao trình độ viết code, thao tác trên phần mềm Unity nhanh hơn.

6. Hướng phát triển

Tối ưu hoá, nâng cấp trò chơi: thêm chức năng chọn nhiều nhân vật, nâng cấp chỉ số sức mạnh cho player, thêm map và các kẻ địch, cạm bẫy đa dạng hơn. Tiếp tục nghiên cứu và tìm hiểu thêm để hoàn thiện các chức năng và tài nguyên còn thiếu sót.

TÀI LIỆU THAM KHẢO

Tài liệu tham khảo bằng tiếng anh.

[1] Basu, S.(2017), *2D PLATFORM GAME: Developed using Unity game engine*, ResearchGate, Springer.

[2] DeWitte, M.(2020), *Unity 2020 by Example Third Edition: A Project-based Guide to Creating 2D, 3D, and VR Games*, Birmingham, Packt Publishing.

[3] Lanzinger, F.(2022), *2D Game Development with Unity*, Boca Raton, CRC Press.

[4] McDonald, P.(2024), *Run and Jump: The Meaning of the 2D Platformer,* Cambridge, MIT Press.

[5] Menard, M. & Wagstaff, B.(2012), *Game Development with Unity, Second Edition*, Boston, Cengage Learning.

[6] Wang, K.(2025), *Leveraging Unity for 2D Pixel Game Development: Techniques and Best Practices*, ResearchGate, Elsevier.