TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA CÔNG NGHỆ THÔNG TIN

2024

THÁI DUY QUÝ, TẠ HOÀNG THẮNG, ĐOÀN MINH KHUÊ

LẬP TRÌNH GAME 2D VỚI UNITY C#

BẢN THẢO
SỐ 1

LỜI NÓI ĐẦU

Chào mừng đến với cuốn sách "Lập trình Game cơ bản với Unity C#"!

Ngày nay, với sự phát triển mạnh mẽ của các thiết bị điện tử và giải trí, song song với đó là sự phát triển của ngành công nghiệp Game. Ngành này đã mang lại một lợi nhuận khổng lồ và có nhiều tiềm năng để phát triển trong tương lai sắp tới.

Việc tạo ra trò chơi không chỉ là việc thách thức sự sáng tạo của bạn mà còn mở ra một thế giới đầy kỳ diệu của công nghệ và lập trình. Có rất nhiều công cụ hỗ trợ cho lập trình Game, tuy nhiên Unity đã trở thành một nền tảng mạnh mẽ cho việc phát triển game, và ngôn ngữ lập trình C# là công cụ dễ học, dễ hiểu để biến ý tưởng thành hiện thực.

Cuốn sách này được thiết kế để hướng dẫn bạn đọc từ những khái niệm cơ bản nhất đến những kỹ thuật phức tạp hơn trong việc xây dựng trò chơi sử dụng Unity và ngôn ngữ lập trình C#. Bất kể bạn là người mới bắt đầu hay đã có kinh nghiệm, cuốn sách này sẽ hỗ trợ bạn từng bước một để tiếp cận và thăng tiến trong thế giới phức tạp của lập trình game.

Qua từng chương, chúng ta sẽ khám phá cách sử dụng Unity để tạo ra cảnh, thêm hình ảnh và âm thanh, xử lý logic game và tương tác người chơi thông qua C#. Tôi hy vọng cuốn sách sẽ giúp bạn không chỉ hiểu rõ quy trình tạo game mà còn thúc đẩy sự sáng tạo của bạn.

Hãy sẵn sàng cho cuộc phiêu lưu hấp dẫn này và bắt đầu hành trình lập trình game với Unity và C#!

MỤC LỤC

Chương 1. Tổng quan về lập trình game

Lập trình game là một trong những lĩnh vực phát triển phức tạp và thú vị nhất trong ngành công nghiệp giải trí. Lĩnh vực này giúp tạo ra các trải nghiệm thú vị, tương tác và thậm chí là thế giới mới thông qua thiết bị cầm tay, màn hình máy tính hoặc thiết bị di động của bạn. Trong chương này, chúng ta sẽ làm quen với thế giới lập trình game và vai trò quan trọng của lĩnh vực này trong đời sống.

1. Lịch sử của lập trình Game

Lập trình game đã trải qua một hành trình dài và phát triển từ những ngày đầu của máy tính cho đến ngày nay. Dưới đây là một số điểm cốt lõi trong tiến trình này:

Thập kỷ 1950 và 1960: Xuất hiện các trò chơi đầu tiên trên máy tính lớn với mục đích giải trí và giáo dục như Tennis for Two, Spacewar…

Thập kỷ 1970: Khởi nguồn của ngành công nghiệp lập trình game với sự ra đời của Atari và các trò chơi Arcade hay trò chơi kinh điển như Pong.

Thập kỷ 1980: Với sự phát triển vượt bậc của hệ thống phần cứng, bắt đầu xuất hiện của các hệ thống game như Nintendo Entertainment System (NES), Super Mario Bros, Tetris.

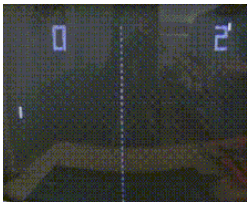
Thập kỷ 1990: Sự phát triển của công nghệ máy tính cá nhân dẫn đến sự tăng cường và phổ biến của các game trên PC như sự ra đời của các dòng game dạng RPG - Role-Playing Game (Final Fantasy, Doom).

Thập kỷ 2000: Sự xuất hiện, phổ biến và phát triển của điện thoại di động và Internet đã dẫn tới sự ra đời của nhiều thể loại trò chơi, thay đổi cách chúng ta trải nghiệm trò chơi. Có rất nhiều nền tảng game đã phát triển trên di động và trực tuyến, mang lại một cuộc cách mạng giải trí mới cho mọi người.

Hiện nay: Ngoài sự phát triển của các dòng game giải trí, người ta đang hướng tới game tích hợp công cụ trí tuệ nhân tạo (AI), game trên BlockChain và game trên Vũ trụ ảo (Metaverse)

Tennis for Two (1950s)

Spacewar (1960s)



Pong (1970s)

Super Mario Bros (1980s)

Doom (1990s)

FIFA game (2000s)

Mobile games (2000s)

Metaverse games (2022)

Hình 1.1. Một số hình ảnh game phát triển theo thời gian (Nguồn: Internet)

2. Khái niệm cơ bản về trò chơi

Một trò chơi được tạo nên bởi các yếu tố chính như mục tiêu, luật lệ, hệ thống phản hồi và sự yêu thích của người chơi.

Mục tiêu: Mục tiêu chính là đích cần đạt đến của trò chơi, những người tham gia sẽ cố gắng để đạt được mục tiêu đó. Ví dụ như trong trò chơi bóng đá, các cầu thủ sẽ cố gắng để đạt được mục tiêu đó là ghi bàn vào lưới đối phương.

Luật lệ: Các trò chơi sẽ luôn được thiết kế dựa trên các luật lệ. Các luật lệ này sẽ làm cho việc hoàn thành mục tiêu của người chơi trở nên khó hơn, đồng thời cũng là điều tạo ra sự hấp dẫn của mỗi trò chơi. Ví dụ như trong Flappy Bird, người chơi có mục tiêu điều khiển con chim bay càng xa càng tốt, tuy nhiên với điều luật không được để chim chạm vào ống nước.

Đối tượng chính (Player): Đối tượng chính là chủ thể mà người chơi sẽ sử dụng, điều khiển thể khám phá hệ thống cấp bậc của Game. Ví dụ một chàng hiệp sĩ, một anh thợ sửa ống nước, một chiếc máy bay… Đối tượng chính sẽ được tác động bởi 2 thuộc tính là điểm bắt đầu và điểm kết thúc trò chơi.

Đối tượng phản diện (Enemy): Đối tượng phản diện là nguyên nhân trực tiếp hoặc gián tiếp khiến người chơi không thể vượt qua màn chơi. Trong một số thiết kế màn chơi, đối tượng phản diện là yếu tố chính để cấu thành nên tư duy về thiết kế độ khó. Ví dụ quái vật, trái bom, hay một chiếc xe chạy ở chiều ngược lại.

Đối tượng môi trường: Đối tượng môi trường là vùng không gian và thời gian để đối tượng chính và đối tượng phản diện hoạt động. Các đối tượng môi trường có thể có tác động đến hai đối tượng trên. Cũng như nhận lại sự tác động của chúng. Có thể áp dụng các đối tượng môi trường để gián tiếp tạo nên độ khó cho màn chơi. Ví dụ như mặt đất, các chướng ngại vật (thác nước, hố gai, bụi rậm…) hay các cổng teleport, các trigger…

Nếu các đối tượng phản diện được xem như công cụ hữu hiệu để thể hiện "độ sâu" của thiết kế màn chơi. Thì các đối tượng môi trường có thể dùng để thể hiện "độ rộng" của màn chơi cũng như nội dung game. Trong một số trường hợp cá biệt, một số đối tượng môi trường cũng có thể mang một số đặc tính như một đối tượng phản diện, nhưng ở thể thụ động hơn.

Hệ thống phản hồi: Hệ thống phản hồi là sự tương tác giữa trò chơi và người chơi, giúp họ biết họ đang cách mục tiêu của mình bao xa. Nó có thể ở dạng điểm, cấp độ... Trong các trò chơi điện tử, hệ thống này còn được thiết kế thêm để giúp người chơi biết được các hành động của mình đang được ghi nhận như thế nào. Ví dụ như khi bấm nút sẽ có âm thanh kèm theo, hoặc khi bị mất máu sẽ có nhấp nháy màu đỏ trên màn hình. Một trò chơi với hệ thống phản hồi chi tiết sẽ mang lại cảm giác cho người chơi tốt hơn.

Sự yêu thích của người chơi: Để một trò chơi thực sự tồn tại, nó phải có sự tham gia của người chơi, và hơn hết nó phải có được sự yêu thích từ khán giả. Điều này yêu cầu những nhà sản xuất phải giúp người chơi hiểu được mục tiêu, nắm rõ luật Lệ và cảm nhận được hệ thống phản hồi. Đó là lý do mà chúng ta thường thấy các trò chơi điện tử hiện nay có những cốt truyện thú vị, hình ảnh bắt mắt, hướng dẫn cho người chơi mới vô cùng cẩn thận và luôn tìm cách giữ chân họ bằng nhiều sự kiện hấp dẫn.

Hệ thống thiết kế màn chơi: Hệ thống thiết kế màn chơi có thể gọi nôm na là tạo các bộ thư viện dữ liệu của các mô đun chính. Quy trình này có thể xoay quanh việc trả lời các câu hỏi được đặt ra cho các đối tượng.

Thiết kế màn chơi là một công việc khá thú vị và cũng mang đầy tính sáng tạo. Nếu xây dựng các hệ thống vững chắc từ đầu thì gần như khả năng mở rộng các màn chơi của bạn là vô hạn. Ở các cấp độ Thiết kế Game cao hơn, khi bạn đầu tư vào việc thiết kế màn chơi. Về lâu dài, ở bản thân bạn sẽ tự phát triển những kĩ năng bổ trợ rất mạnh để xây dựng nội dung game (tâm lý học, cốt truyện, thiết kế hiệu ứng…). Tất cả những điều đó đều bắt đầu từ những kiến thức cơ bản ban đầu.

3. Các vai trò chính trong quy trình sản xuất game

Quy trình sản xuất game chuyên nghiệp là một quá trình phức tạp và đòi hỏi sự cộng tác của nhiều vai trò khác nhau để cho ra một trò chơi hoàn chỉnh và chất lượng đến với người dùng. Dưới đây là một số vai trò chính trong quy trình sản xuất game thường thấy ở các công ty hiện nay:

Nhà phát triển (Developers): Nhà phát triển là những người sáng tạo ra trò chơi, bao gồm các lập trình viên, thiết kế đồ họa, nhà thiết kế trò chơi, và các chuyên gia khác. Họ chính là người tạo ra mã nguồn, đồ họa, âm thanh và nội dung chơi của trò chơi.

Nhà thiết kế trò chơi (Game Designers): Nhà thiết kế trò chơi đảm nhận trách nhiệm tạo ra cấu trúc và quy tắc chơi của trò chơi. Họ thiết kế cơ chế chơi, cấu trúc cốt truyện, và tạo ra các thử thách và nhiệm vụ cho người chơi.

Nghệ sĩ đồ họa (Artists): Nghệ sĩ đồ họa tạo ra các hình ảnh, mô hình 3D, và thiết kế các môi trường và nhân vật trong trò chơi. Những người này đảm bảo rằng trò chơi có một hình ảnh hấp dẫn và có tính thẩm mỹ.

Lập trình viên (Programmers): Lập trình viên viết mã nguồn để triển khai các chức năng và tính năng trong trò chơi. Họ làm việc với các công cụ và ngôn ngữ lập trình để tạo ra trò chơi hoạt động một cách mượt mà và ổn định.

Nhà làm phim (Animators): Nhà làm phim tạo ra các hoạt ảnh và chuyển động cho nhân vật và các đối tượng trong trò chơi. Họ đảm bảo rằng các phần tử trong trò chơi di chuyển và tương tác một cách tự nhiên và hấp dẫn.

Nhà thiết kế âm thanh (Sound Designers): Nhà thiết kế âm thanh tạo ra âm thanh và nhạc nền cho trò chơi, giúp tạo ra không gian âm thanh ấn tượng và tương tác âm thanh thích hợp.

Nhà sản xuất (Producers): Nhà sản xuất giúp quản lý dự án và đảm bảo rằng trò chơi được phát triển đúng theo kế hoạch và ngân sách. Họ có trách nhiệm lập kế hoạch, quản lý tài nguyên và tiến độ sản xuất.

Kiểm tra chất lượng (Quality Assurance - QA): Nhóm kiểm tra chất lượng thử nghiệm trò chơi để xác định lỗi và vấn đề liên quan đến trải nghiệm người chơi. Nhóm này đảm bảo rằng trò chơi hoạt động một cách mượt mà và không có lỗi kỹ thuật.

Nhà xuất bản (Publishers): Nhà xuất bản là các công ty hoặc tổ chức có trách nhiệm phát hành và quảng cáo trò chơi. Họ cung cấp tài chính và hỗ trợ tiếp thị để đưa trò chơi ra thị trường.

Biên kịch (Writers): Biên kịch viết cốt truyện, văn bản và hội thoại cho trò chơi, giúp tạo ra một thế giới và câu chuyện sâu sắc và thu hút người chơi.

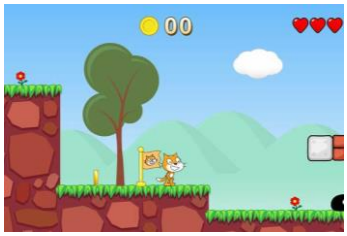
Mỗi vai trò trong quá trình sản xuất game đóng góp một phần quan trọng để tạo ra một trò chơi thành công và thú vị cho người chơi. Các thành viên trong đội ngũ làm việc cùng nhau để đảm bảo rằng trò chơi hoàn thiện đáp ứng các tiêu chuẩn chất lượng và thị trường.

4. Các thể loại game

Các thể loại game được chia thành nhiều loại dựa trên nhiều yếu tố như lối chơi, mục tiêu, phong cách, hình ảnh, nền tảng phân phối.

Nếu chia theo hình ảnh, có thể chia thành các loại game như: 3D game, 2D game, Pixel Art game… Nếu chia theo lối chơi và loại mục tiêu, có thể chia thành các loại như Hành động (Action), Nhập vai (Role-Playing), Chiến thuật (Strategy), Giả tưởng (Fantasy), Thể thao (Sports), Mô phỏng (Simulation), Trí tuệ (Puzzle), Đối kháng (Fighting)… Nếu chia theo loại người chơi tham gia có thể có các loại game như Casual (thông thường), Hyper Casual (Game phổ thông) và Mid/Hard Core (Hạng nặng). Hoặc nếu chia theo hình thức game, có thể có các loại game như game Giải trí, AI game, BlockChain game…


3D Game


2D Game


Pixel Art Game


Action Game


Sport Game


Puzzle Game


Casual Game


Hyper Casual Game


Mid/Hard Core Game

Hình 1.2. Một số hình ảnh thể loại game (Nguồn: Internet)

5. Các yếu tố trong lập trình game

Lập trình game đòi hỏi nhiều yếu tố kết hợp với nhau. Sau đây là các yếu tố chính trong lập trình game:

Đồ họa (Graphics): Đồ họa là một phần quan trọng của trò chơi và bao gồm việc tạo ra hình ảnh và mô hình cho nhân vật (2D hoặc 3D), môi trường và các đối tượng trong trò chơi. Đồ họa được tạo ra bằng cách sử dụng các công cụ và thư viện đồ họa để vẽ, tạo mô hình, và làm hiệu ứng hình ảnh.

Âm thanh (Sound): Âm thanh góp phần quan trọng để tạo ra trải nghiệm trò chơi hấp dẫn. Lập trình viên âm thanh tạo ra âm thanh hiệu ứng, nhạc nền, và sử dụng các kỹ thuật chế tác âm thanh để tạo ra các hiệu ứng phù hợp.

Điều khiển và tương tác (Input and Interaction): Người chơi tương tác với trò chơi thông qua bàn phím, chuột, hoặc bộ điều khiển. Lập trình viên phải xử lý và đáp ứng các tương tác người chơi để điều khiển nhân vật và tương tác với thế giới trong trò chơi.

Vật lý (Physics): Các trò chơi thường có các yếu tố vật lý, chẳng hạn như trọng lực, va chạm hoặc gắn kết lò xo. Các máy làm game thường tạo ra mô phỏng vật lý sẵn để xử lý các hiệu ứng này trong trò chơi.

Cơ chế trò chơi (Game Mechanics): Cơ chế trò chơi là các quy tắc và hành vi mà trò chơi tuân theo, bao gồm quy tắc chơi, cấu trúc nhiệm vụ và hệ thống tiến hóa nhân vật. Lập trình viên cài đặt và quản lý các cơ chế này để đảm bảo trò chơi hoạt động một cách mượt mà và thú vị.

Kỹ thuật mạng (Networking): Trò chơi đa người chơi yêu cầu kỹ thuật mạng để kết nối và tương tác giữa các người chơi qua Internet. Lập trình viên mạng phải quản lý kết nối, đồng bộ hóa dữ liệu và giải quyết các vấn đề liên quan đến độ trễ và độ ổn định.

Lập trình trí tuệ nhân tạo (Artificial Intelligence - AI): Trong trò chơi đơn người hoặc đa người, AI quản lý hành vi của các nhân vật không người chơi hoặc đối thủ máy tính. Lập trình viên AI phải tạo ra các thuật toán và quyết định để làm cho NPCs hoạt động một cách thông minh và có thể thách thức người chơi.

Quản lý tài nguyên (Resource Management): Trò chơi thường phải quản lý tài nguyên như bộ nhớ, CPU, và GPU để đảm bảo trò chơi chạy mượt mà trên các thiết bị khác nhau. Lập trình viên phải tối ưu hóa mã nguồn để sử dụng tài nguyên hiệu quả.

Lập trình đa nền tảng (Cross-Platform Development): Để trò chơi có thể chạy trên nhiều nền tảng (PC, console, di động), lập trình viên phải sử dụng các công cụ và kỹ thuật đa nền tảng.

Giao diện người dùng (User Interface - UI): Lập trình viên UI tạo ra giao diện người dùng để người chơi có thể tương tác với trò chơi, chẳng hạn như các menu, hộp thoại, và chỉ số.

Cơ sở dữ liệu (Database): Trò chơi có thể sử dụng cơ sở dữ liệu để lưu trạng thái trò chơi, tiến trình và dữ liệu người chơi. Lập trình viên cơ sở dữ liệu xây dựng và quản lý các cơ sở dữ liệu này.

Tất cả các yếu tố này kết hợp lại để tạo ra một trò chơi hoàn chỉnh và thú vị. Lập trình viên game phải có kiến thức về nhiều lĩnh vực khác nhau và làm việc chặt chẽ với các thành viên trong đội phát triển để đảm bảo rằng trò chơi được triển khai thành công.

6. Các công cụ phát triển game

Có nhiều công cụ và môi trường phát triển game khác nhau mà các nhà phát triển có thể sử dụng để tạo ra các trò chơi từ các thể loại khác nhau. Bảng 1.1 minh họa một số công cụ chính dùng để phát triển game.

Bảng 1.1. Một số công cụ phát triển game chuyên dụng

STT	Tên phần mềm và nhà sản xuất	Chi phí	Ngôn ngữ lập trình	Nền tảng hỗ trợ	Game nổi bật
1	Ulreal Engine -> *Epic Games*	Miễn phí sử dụng ban đầu	C++, Python	Windows, Mac, Linux, iOS, Android, Playstation, Xbox.	Marvel Heroes, Infinity Blade, Fortnite Battle Royale

2	Unity Engine ->*Unity Technologies*	Miễn phí cá nhân. Thu phí cho bản Plus và Pro	C#, JavaScript hoặc Boo	Windows, Mac, iOS, Android, Playstation, Xbox, Windows Phone	Pokémon GO, Super Mario Run, Angry Birds 2, Wasteland 2, For Honor
3	Cocos2d-x MIT License	Miễn phí	C++, LUA và JavaScript	iOS, Android, Windows Phone, Tizen, Windows, Mac, Linux, Xbox	Big Fish Casino, Dragon City, BADLAND, Castle Clash
4	Construct Scirra Ltd	Miễn phí; Thu phí cho bản Pro	HTML5, CSS3 và JavaScript	Android, iOS, PC, Console, Html5	Guinea Pig Parkour, Small Saga, Last moon, Creature keeper
5	GameMaker Studio YoYo Games	Có phí	C++, C#	Android, iOS	Undertale, KatanaZero, Hiper Light Drifter, Nuclear Throne

Ngoài ra, có nhiều thư viện lập trình như libGDX (Java), Pygame (Python), SDL (C/C++), HTML5/CSS3, Scratch… được sử dụng để phát triển game. Các thư viện này thường gắn liền với một ngôn ngữ lập trình để giúp người dùng vận dụng được thế mạnh và tính quen thuộc của ngôn ngữ lập trình đó.

Trong các công cụ được giới thiệu ở trên thì Unity là một trong những công cụ lập trình Game phổ biến nhất. Ở thời điểm hiện tại (năm 2023), 34% trong số 1,000 mobile game miễn phí được sản xuất bằng Unity. Unity Engine phục vụ cho cả các Artist, Designer và Developer có thể cùng nhau phát triển trò chơi của mình với nội dung, hình ảnh tuyệt đẹp, hỗ trợ các công cụ thiết kế cả 2D và 3D. Hệ thống hỗ trợ xây dựng Animation của Unity cũng cực kỳ mạnh mẽ. Nếu như Unreal Engine phổ biến hơn với các trò chơi trên PC thì Unity lại vô cùng phù hợp hơn với các trò chơi di động. Các game được sản xuất bằng Unity có dung lượng nhẹ hơn, chạy ổn định dù sử dụng đồ họa 3D. Không những thế, Unity hiện tại hỗ trợ xây dựng cả các trò chơi VR và AR một cách dễ dàng.

7. Các nền tảng phân phối game

Khi đã phát triển và hoàn thành một trò chơi, chúng ta có nhiều tùy chọn để phân phối và xuất bản đến người chơi. Sau đây là một số nền tảng phân phối game phổ biến hiện nay:

Cửa hàng ứng dụng di động: App Store (iOS) dùng để phân phối game cho các thiết bị Apple như iPhone và iPad, trong khi đó Google Play Store (Android) lại dành cho các thiết bị chạy hệ điều hành Android.

Cửa hàng game trên PC: Steam là một trong những cửa hàng game trực tuyến lớn nhất cho PC; Epic Games Store cũng là cửa hàng game trực tuyến cạnh tranh với Steam; Trong khi đó GOG (Good Old Games) được biết đến với việc phân phối game cổ điển và game DRM-free.

Các nền tảng trò chơi kỹ thuật số: Xbox Live dung cho các loại game trên các thiết bị Xbox của Microsoft; PlayStation Network (PSN) dùng để phân phối game trên các thiết bị PlayStation của Sony; Nintendo eShop thì dành cho các game trên các hệ máy chơi game Nintendo.

Nền tảng trực tuyến độc lập: Một số nền tảng giúp phân phối các game độc lập như itch.io, GameJolt, Oculus Store, Humble Bundle hoặc GameSpot.

Việc Lựa chọn nền tảng phụ thuộc vào mục tiêu của nhà phát triển, ngân sách, loại trò chơi và đối tượng mục tiêu. Một số nhà phát triển chọn phát triển game cho nhiều nền tảng để tiếp cận nhiều người chơi hơn.

8. Kết chương

Chương này đã giúp bạn hiểu về một số khái niệm chính trong lập trình game, cơ hội và thách thức trong lĩnh vực lập trình game. Trong chương tiếp theo, chúng ta sẽ tìm hiểu chi tiết về công cụ lập trình game Unity.

Bài tập

1. Hãy nêu các thời kỳ trong lịch sử phát triển game. Các công nghệ game đã phát triển và tiến bộ như thế nào qua các thời kỳ này

2. Trong game thì người chơi và đối thủ có vai trò gì?

3. Hãy nêu các vai trò chính trong quy trình sản xuất game. Nếu một lập trình viên đi theo lĩnh vực game thì sẽ thực hiện ở vai trò nào là tốt nhất?

4. Hãy cho biết các thể loại game và cho ví dụ về từng thể loại.

5. Hãy cho biết các công cụ phổ biến dùng để lập trình game hiện nay. Vì sao trong các công cụ đó thì Unity mặc dù mới nhưng lại đang nổi lên như một công cụ phổ biến?

Chương 2. Tổng quan phần mềm Unity

Trong các phần mềm dùng để lập trình Game thì Unity đang nổi lên như một phần mềm khá phổ biến, có nhiều cộng đồng hỗ trợ. Chương này sẽ giúp độc giả hiểu rõ về cách cài đặt cũng như vận hành một số chức năng chính trong Unity.

1. Giới thiệu chung

Unity là một trong những công cụ phát triển game phổ biến nhất và mạnh mẽ. Một số thông số kỹ thuật về Unity như sau:

Đa nền tảng: Unity cho phép bạn phát triển game cho nhiều nền tảng khác nhau, bao gồm PC, Mac, Linux, iOS, Android, PlayStation, Xbox, Nintendo Switch và nhiều nền tảng khác. Điều này giúp nhà phát triển tiếp cận một lượng lớn người chơi trên nhiều thiết bị khác nhau.

Ngôn ngữ lập trình: Unity hỗ trợ ngôn ngữ lập trình C#, một ngôn ngữ phổ biến trong ngành phát triển game. C# có cú pháp dễ đọc và mạnh mẽ, cho phép dễ dàng tạo ra các tính năng và gameplay phức tạp.

Thư viện và Asset Store: Unity asset store có một cộng đồng lớn và sôi động, cung cấp nhiều tài nguyên miễn phí và trả phí. Nhà phát triển có thể tải xuống và sử dụng các prefab, mẫu, hình ảnh, âm thanh, và nhiều tài nguyên khác để giúp bạn phát triển game nhanh hơn.

Môi trường phát triển tích hợp (IDE): Unity cung cấp một môi trường phát triển tích hợp (IDE) cho việc lập trình và thiết kế game. Bạn có thể dễ dàng chỉnh sửa mã, xây dựng và xem kết quả ngay trong môi trường Unity hoặc thông qua các công cụ khác như Visual Studio, Visual Studio Code.

Đồ họa và âm thanh: Unity hỗ trợ đồ họa 2D và 3D, cho phép bạn tạo ra các trò chơi 2D đơn giản hoặc trò chơi 3D phức tạp. Nó cũng có tích hợp âm thanh và hỗ trợ cho cả âm thanh 2D và 3D.

Hỗ trợ thực tế ảo và thực tế ảo tăng cường: Unity cung cấp hỗ trợ cho phát triển ứng dụng thực tế ảo (VR) và thực tế ảo tăng (AR) thông qua các plugin và tích hợp.

Cộng đồng và tài liệu: Unity có một cộng đồng lớn và nhiều tài liệu học tập trực tuyến. Nhà phát triển có thể tìm kiếm câu hỏi, hướng dẫn và tài liệu trực tuyến để giải quyết các vấn đề phát triển game của họ. Đồng thời Unity cũng cho phép nhúng và tích hợp các công cụ bên ngoài như 3DS Max, Maya, Photoshop và nhiều công cụ khác vào quy trình làm việc trong Unity.

2. Cài đặt Unity

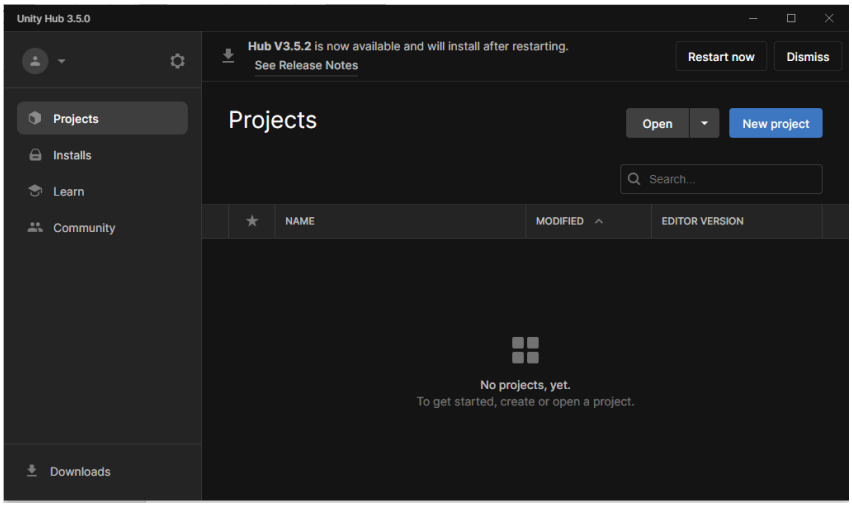
Để cài đặt Unity từ phiên bản 2020 trở lại đây, cần cấu hình máy tính RAM tối thiểu 8GB, ổ cứng dung lượng còn lại 20GB; Bên cạnh đó, để thuận tiện hơn trong quá trình lập trình, bạn cần cài đặt thêm công cụ Visual Studio nên cần thêm dung lượng ổ đĩa cứng để cài công cụ này. Quá trình cài đặt Unity phải thông qua một công cụ quản lý, đó là Unity Hub. Quá trình cài đặt theo các bước như sau:

Bước 1: Tải Unity Hub

Unity Hub là một ứng dụng quản lý dự án Unity và tất cả các phiên bản Unity bạn cài đặt. Bạn cần tải và cài đặt Unity Hub trước khi cài đặt Unity. Bạn có thể tải Unity Hub từ trang web chính thức của Unity:

https://unity.com/releases/editor/archive

Unity Hub sẽ quản lý các phiên bản Unity, bản quyền, tài khoản. Unity Hub cũng giúp cho việc tạo mới và mở các dự án của Unity sau này.



Hình 2.1. Giao diện Unity Hub

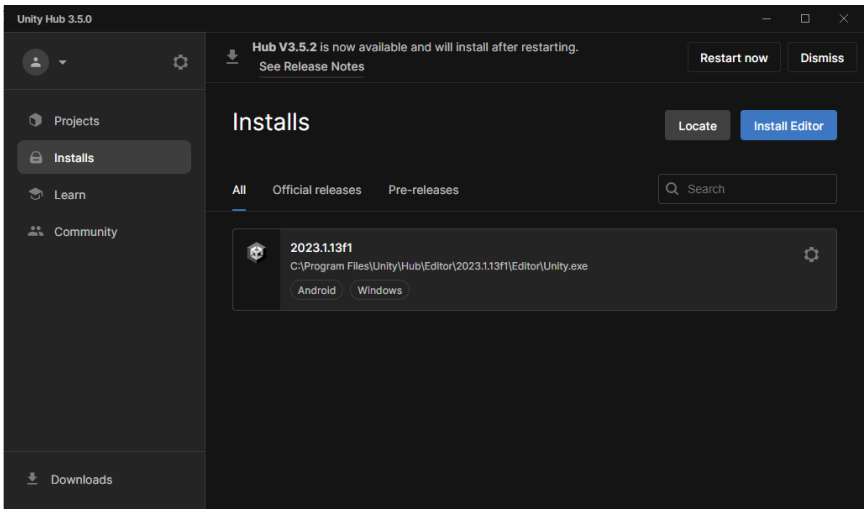
Bước 2: Tải Unity

Sau khi bạn đã cài đặt Unity Hub, bạn có thể sử dụng nó để tải và cài đặt các phiên bản Unity mà bạn muốn. Sau đây là cách thực hiện:

- Mở Unity Hub, nhấp vào tab “Installs” (Cài đặt);

- Nhấp vào nút “Add” (Thêm). Chọn phiên bản Unity bạn muốn cài đặt. Bạn cũng có thể tùy chỉnh các thành phần cài đặt tùy chọn nếu cần. Nhấp vào nút “Next”, chọn nơi bạn muốn cài đặt Unity và các dự án của mình. Nhấp vào nút “Done” để bắt đầu quá trình cài đặt.

Sau khi cài đặt xong, có thể kiểm tra các bản cài đặt tại phần Installs bên trái:



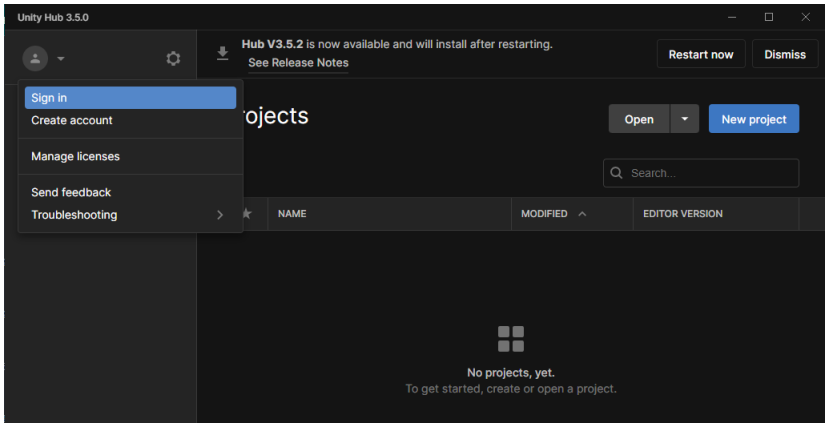
Hình 2.2. Giao diện Unity Hub chứa các phiên bản cài đặt

Bước 3: Đăng ký hoặc đăng nhập vào tài khoản Unity

Để sử dụng Unity, bạn cần có một tài khoản Unity. Bạn có thể đăng ký tài khoản mới hoặc đăng nhập vào tài khoản hiện có qua Unity Hub. Thao tác đăng ký và đăng nhập có thể dễ dàng thực hiện trong Unity Hub bằng cách nhấp chuột vào nút hình nhân ở góc trên, bên trái (Hình 2.3). Unity cho phép đăng nhập thông qua tài khoản Google hoặc Facebook.

Bước 4: Kích hoạt cấp phép Unity

Thông thường, người dùng mới sẽ được Unity hỗ trợ cấp phép dùng phiên bản miễn phí 1 năm với các tính năng ổn định và hiệu quả. Để được cấp phép, sau khi đăng nhập, người dùng sẽ kích hoạt bằng cách bấm chọn vào nút hình nhân ở góc trên, bên trái, chọn Manage licenses và khai báo các thông tin cho Unity. Thời gian sử dụng là một năm nhưng khi hết hạn có thể kích hoạt lại.



Hình 2.3. Giao diện cho phép đăng nhập và kích hoạt bản quyền

Với gói Unity Pro, người dùng cũng sẽ cần kích hoạt nó thông qua Unity Hub sau khi cài đặt. Unity sẽ hướng dẫn nhập thông tin qua quy trình kích hoạt cấp phép.

Bước 5: Sử dụng Unity

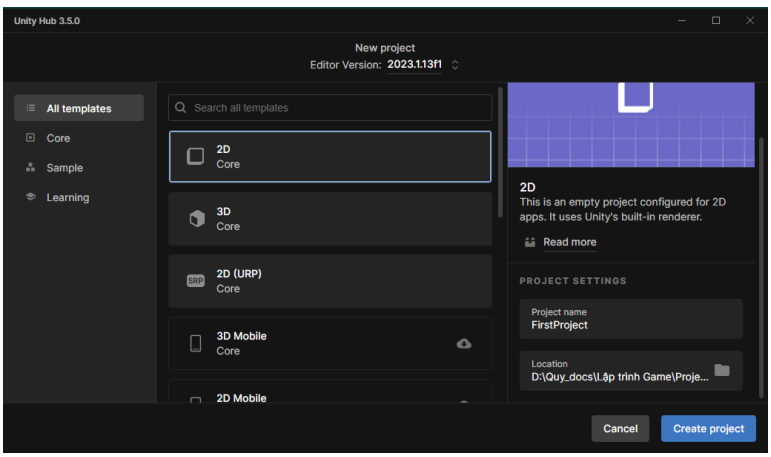
Sau quá trình đăng ký tài khoản và thiết lập bản quyền hoàn tất, bạn có thể mở Unity Hub và bắt đầu sử dụng Unity để phát triển dự án của bạn. Bạn có thể tạo dự án mới, mở các dự án hiện có và bắt đầu làm việc.

3. Tạo dự án Unity

Unity Hub cho phép tạo nhiều loại dự án như 2D (ứng dụng hai chiều), 3D (ứng dụng ba chiều), VR (công nghệ thực tại ảo), AR (công nghệ thực tại tăng cường), Cinematic… Ngoài ra Unity Hub còn cung cấp các phần giúp nhà phát triển nâng cao kỹ năng như Core, Sample (mẫu sẵn có), Learning (Các bài học). Trong cuốn sách này, tác giả sẽ chỉ hướng dẫn tạo dự án với 2D, các phần còn lại độc giả tự tìm hiểu thêm.

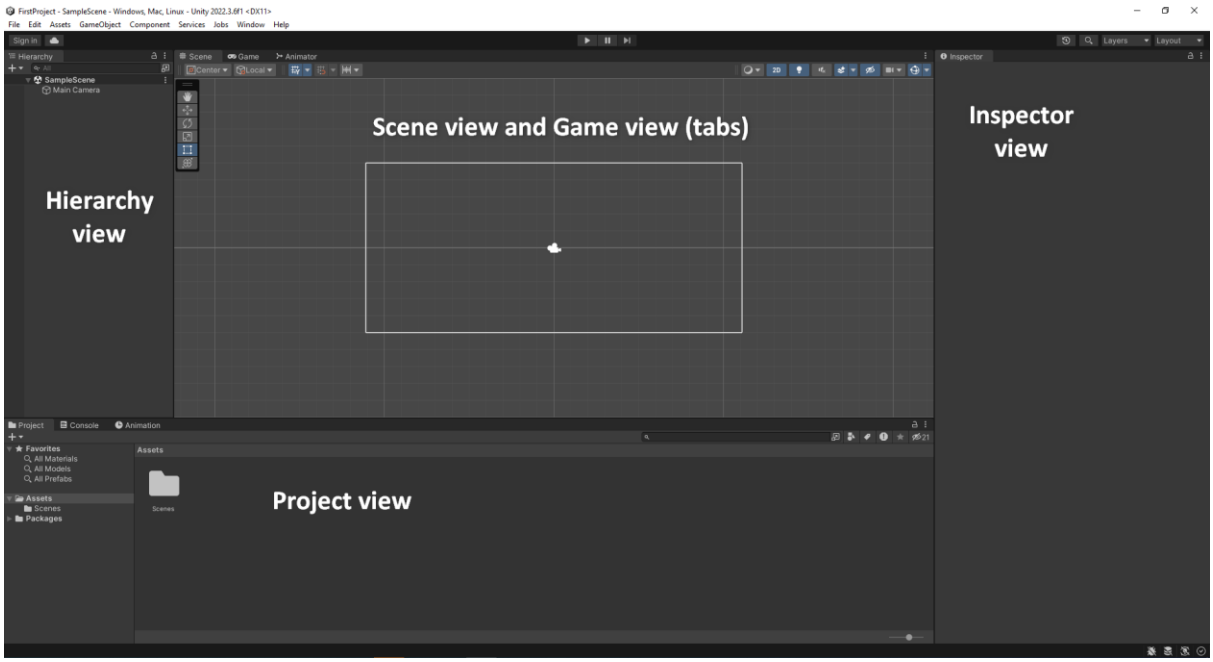
Để tạo dự án với Unity 2D, cần theo các bước như sau:

Bước 1: Mở Unity Hub, chọn 2D Core, đặt tên dự án và nơi lưu trữ trong phần Project Settings, chọn Create project (Hình 2.4)



Hình 2.4. Bắt đầu tạo mới dự án

Bước 2. Unity sẽ tự động tạo ra một dự án đầu tiên với giao diện màn hình như sau. Đây chính là màn hình Scene, dùng để thiết kế các thành phần và đối tượng trong game.

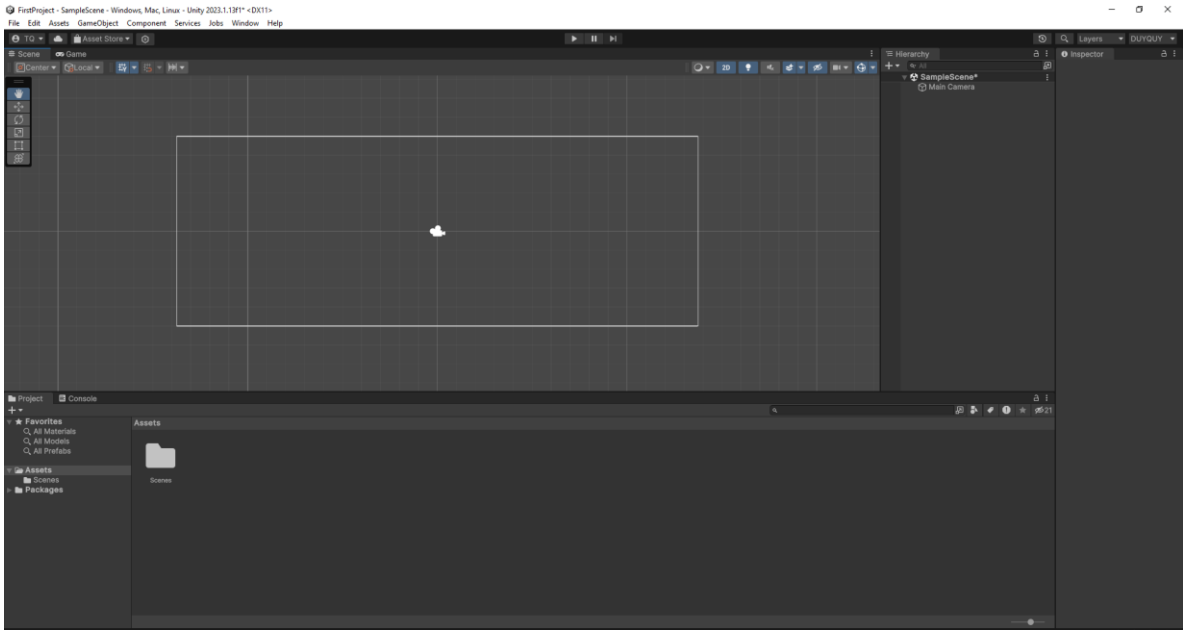


Hình 2.5. Giao diện màn hình Unity mặc định

Bước 3. Bấm vào nút Play () ở trên-giữa màn hình để chạy game, lúc này Unity sẽ chuyển qua màn hình Game. Tất nhiên vì là game mới tạo là game trống nên chỉ là một màn hình màu xanh.

4. Giao diện tổng quan của Unity

Khi bắt đầu tạo mới một dự án, Unity sẽ tự động sắp xếp giao diện theo trật tự như Hình 2.5. Tuy nhiên người dùng có thể thay đổi cách sắp xếp này bằng cách kéo và thả chuột vào các khu vực thích hợp, sau đó vào mục Layout (góc trên, bên phải) để lưu lại cách bố trí giao diện để sử dụng cho lần sau (Hình 2.6).



Hình 2.6. Giao diện mới được lưu và thiết lập với tên DUYQUY

Các thành phần chính của giao diện Unity bao gồm các thành phần như trong Hình 2.5:

- Hệ thống menu: Hệ thống menu cho phép chọn lựa các chức năng phức tạp trong quá trình lập trình và thao tác như: File để quản lý tập tin, Edit để quản lý sự thay đổi, Assets quản lý các tài nguyên, GameObject quản lý các đối tượng trong game, Component quản lý các thành phần, Services quản lý các dịch vụ, Jobs quản lý các tác vụ, Windows quản lý các cửa sổ và Help là phần trợ giúp.

- Các nút chức năng: Hệ thống chức năng bao gồm thanh lựa chọn gồm 4 tính năng: Chọn đối tượng (), Dịch chuyển đối tượng (), Xoay đối tượng (), Phóng to/thu nhỏ đối tượng (), Chọn nhiều đối tượng () và Biến đổi tổng hợp (). Bên cạnh đó, hệ thống chức năng còn bao gồm 4 nút nằm giữa màn hình bao gồm: Play, Pause, và Step dùng để điều khiển game khi cần (). Ngoài ra, còn một số chức năng khác sẽ được giới thiệu kỹ hơn ở chương tiếp theo.

- **Cửa sổ Scene và Game**: Sửa sổ Scene dùng để thiết kế các màn chơi, mỗi màn chơi sẽ bao gồm một Scene. Cửa sổ này mặc định có một Camera được tạo sẵn. Khi thiết kế, Unity không tự lưu mà người dùng cần lưu lại bằng cách nhấn Ctrl + S hoặc vào File, chọn Save. Cửa sổ Game dùng để hiển thị kết quả biểu diễn (khi người dùng nhấn nút Play).

- **Khung nhìn Hierarchy**: Đây là khung nhìn chứa các đối tượng game (Game objects) trong Scene. Đối tượng game chính là các đối tượng chứa trong một Scene như Nhân vật, đối thủ, chướng ngại vật, đèn, hiệu ứng…. Có thể thêm đối tượng bằng cách click phải lên Hierarchy, chọn New, chọn một đối tượng bất kỳ.

- Khung nhìn Inspector: Khung nhìn này giúp quản lý các thành phần (components) của các đối tượng game. Các thành phần này chỉ hiện lên khi người dùng click chuột vào một đối tượng game cụ thể.

- Khung nhìn Project: Đây là khung nhìn để quản lý tài nguyên của game. Các tài nguyên của game bao gồm các thư mục chứa như: Scenes, Scripts, Animations, Audios, Textures, Materials, Prefabs… Mặc định khi tạo, chỉ có thư mục Scenes, sau này chúng ta sẽ tạo thêm các thư mục tương ứng khác. Lưu ý: Khung nhìn này có thể điều chỉnh để hiển thị ở dạng 1 cột hoặc 2 cột bằng cách chọn vào biểu tượng  .

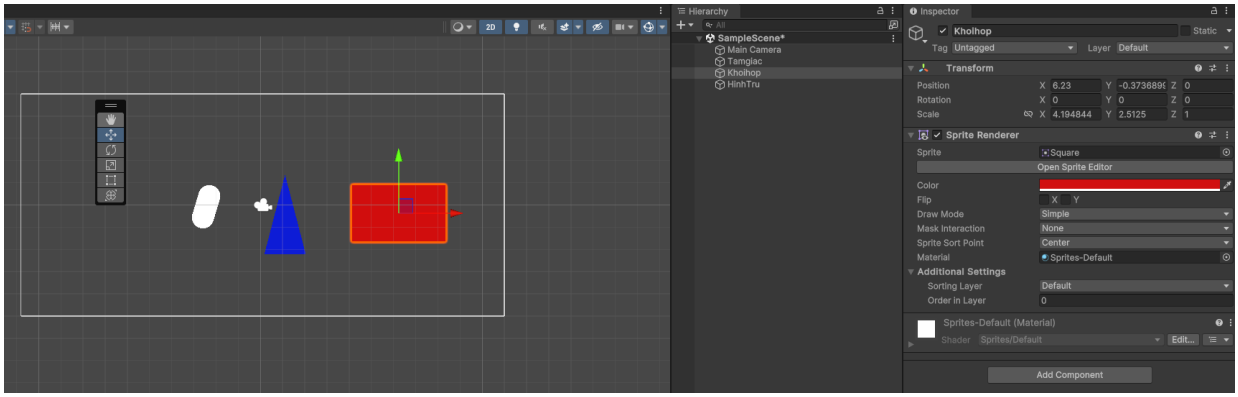
Trong khung nhìn Project còn có thêm cửa sổ Console, đây chính là nơi in ra các dòng lệnh khi người dùng muốn kiểm tra kết quả viết code, hoặc nếu mã nguồn có lỗi, lỗi sẽ được hiện ra ở đây.

5. Một vài thao tác cơ bản

- Thêm 1 hoặc nhiều đối tượng: Click chuột phải lên khung nhìn Hierarchy, chọn New, chọn 2D Objects, chọn Sprites, chọn các đối tượng như Triangle, Square, Circle, Capsule… Các đối tượng này sẽ nằm trên Scene và trên Hierarchy.

- Thay đổi vị trí, kích thước của đối tượng: Trên Hierarchy, dùng chuột chọn từng đối tượng, bấm vào các biểu tượng  , tiến hành dịch chuyển, xoay, co dãn đối tượng và xem kết quả.

- Thay đổi tên, màu sắc của đối tượng: Chọn từng đối tượng, vào khung nhìn Inspector, thiết lập các thông số mới cho X, Y, Z trong phần Transform. Chọn màu sắc trong phần Color của Sprite Renderer và đặt lại tên trong phần đầu của Inpector.



Hình 2.7. Một số thay đổi của đối tượng game khi thay đổi các tham số

Mẹo nhỏ: Khi muốn tìm một đối tượng nào, chỉ cần chọn đối tượng đó trên Hierarchy, đưa chuột ra màn hình Scene, nhấn phím F là Unity sẽ tự tìm đến đối tượng.

6. Một số phím tắt trong Unity

Phần này nêu một số phím tắt cơ bản trong Unity, nếu độc giả dùng máy MacOSX, có thể thay phím Ctrl bằng phím Command.

Bảng 2.1. Một số phím tắt trong Unity (nguồn: www.unity.com)

Tool			
Q	Pan	T	Rect Tool
W	Move	Z	Pivot Mode toggle
E	Rotate	X	Pivot Rotation Toggle
R	Scale	V	Vertex Snap
GameObject			
CTRL/CMD+SHIFT+N		New empty game object	
ALT+SHIFT+N		New empty child to selected game object	
CTRL/CMD+ALT+F		Move to view	
CTRL/CMD+SHIFT+F		Align with view	
SHIFT+F or double-F		Locks the scene view camera to the selected GameObject	
CTRL/CMD+SHIFT+N		New empty game object	
ALT+SHIFT+N		New empty child to selected game object	
CTRL/CMD+ALT+F		Move to view	
CTRL/CMD+SHIFT+F		Align with view	
SHIFT+F or double-F		Locks the scene view camera to the selected GameObject	
Window			
CTRL/CMD+1	Scene	CTRL/CMD+6	Animation
CTRL/CMD+2	Game	CTRL/CMD+7	Profiler

CTRL/CMD+3	Inspector	CTRL/CMD+9	Asset store
CTRL/CMD+4	Hierarchy	CTRL/CMD+0	Version Control
CTRL/CMD+5	Project	CTRL/CMD+SHIFT+C	Console
CTRL/CMD+D	Duplicate	F	Frame (centre) selection
CTRL/CMD+F	Find	CTRL/CMD+P	Play

7. Kết chương

Chương này giúp độc giả có cái nhìn tổng quan về phần mềm Unity, cách cài đặt Unity Hub, cũng như cách vận hành một số chức năng chính. Chương này cũng giới thiệu tổng quan về giao diện và các chức năng cơ bản cũng như một số phím tắt trong Unity.

Bài tập

1. Nêu một số thông số kỹ thuật cho thấy Unity là một trong những công cụ phát triển game phổ biến và mạnh mẽ.

2. Unity Hub là gì? Hãy nêu các gói bản quyền mà Unity xuất bản.

3. Khung nhìn **Hierarchy** khác khung nhìn **Inspector** như thế nào?

4. Hãy nêu sự khác nhau giữa cửa sổ Scene và Game.

5. Tạo một dự án 2D trong Unity, thêm các đối tượng như Hình vuông, hình tròn, tam giác vào trong Sense. Nhấn Play để thấy kết quả.

6. Hãy thực hiện các thao tác như: Thêm 1 hoặc nhiều đối tượng, thay đổi vị trí, kích thước đối tượng, thay đổi tên, màu sắc củ đối tượng. Nhấn Play để thấy kết quả.

Chương 3. Các thành phần cơ bản trong Unity 2D

Sau khi cài đặt thành công, nhiệm vụ của người làm game là phải nắm bắt được các thành phần chính trong Unity. Chương này sẽ giúp người dùng thao tác với một số thành phần cơ bản trong Unity với dự án 2D.

1. Hệ thống Menu

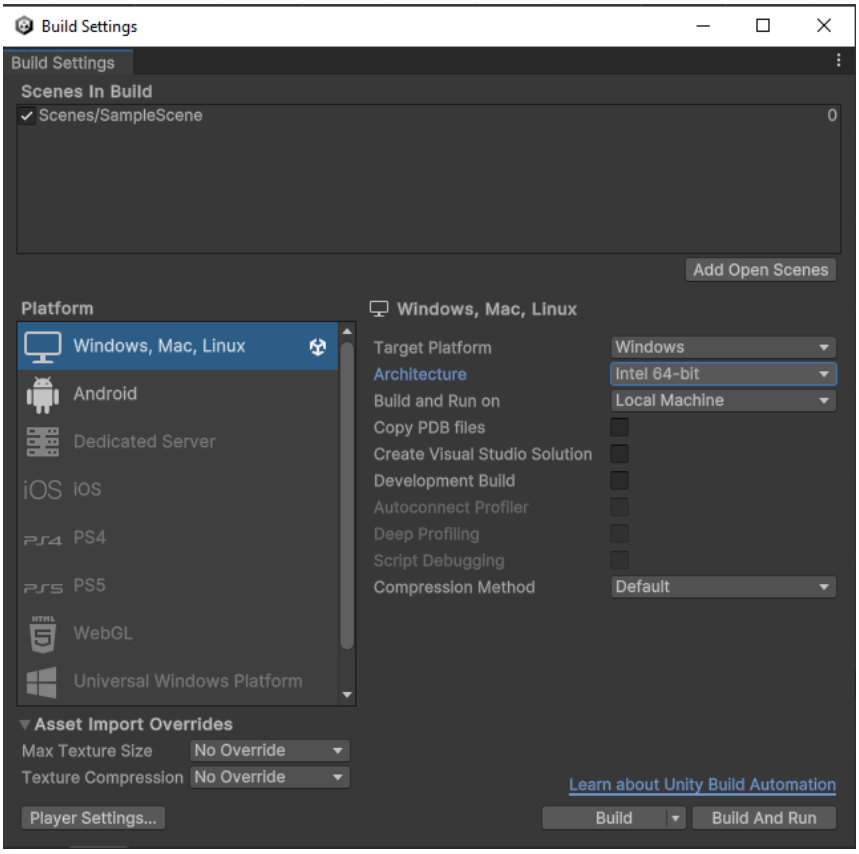
Trong Unity, hầu hết các chức năng có thể thực hiện trên thanh menu. Trong phạm vi sách của cuốn này, tác giả chỉ giới thiệu một số chức năng chính được thể hiện như sau:

Menu File: Chứa các chức năng quản lý tập tin, bao gồm tạo mới và lưu Scene, tạo mới và lưu project. Lưu ý, project chính là toàn bộ dự án game, còn scene là một màn trong game. Khi tạo mới một Scene, nó sẽ được lưu vào thư mục Scenes mặc định của dự án.

Chức năng Build Settings…: Chức năng này cho phép cấu hình dự án trước khi xuất ra tập tin để thực thi:

- Scenes in Build: Chứa các Scene dùng để thực thi, khi cần thực thi Scene nào, chỉ cần kéo Scene đó vào trong ô này, Unity sẽ chỉ định tên và chỉ số dùng để điều khiển code sau này.

- **Platform**: Chứa các nền tảng để thực thi. Unity mặc định thực thi thành tập tin chạy trên Window, Mac, Linux với các cấu hình tương ứng.



Hình 3.1. Giao diện Build Settings

Nếu cần chuyển qua các Platform các như Android hay iOS, chúng ta cần cài đặt thêm gói và nhấn vào nút Switch Platform.

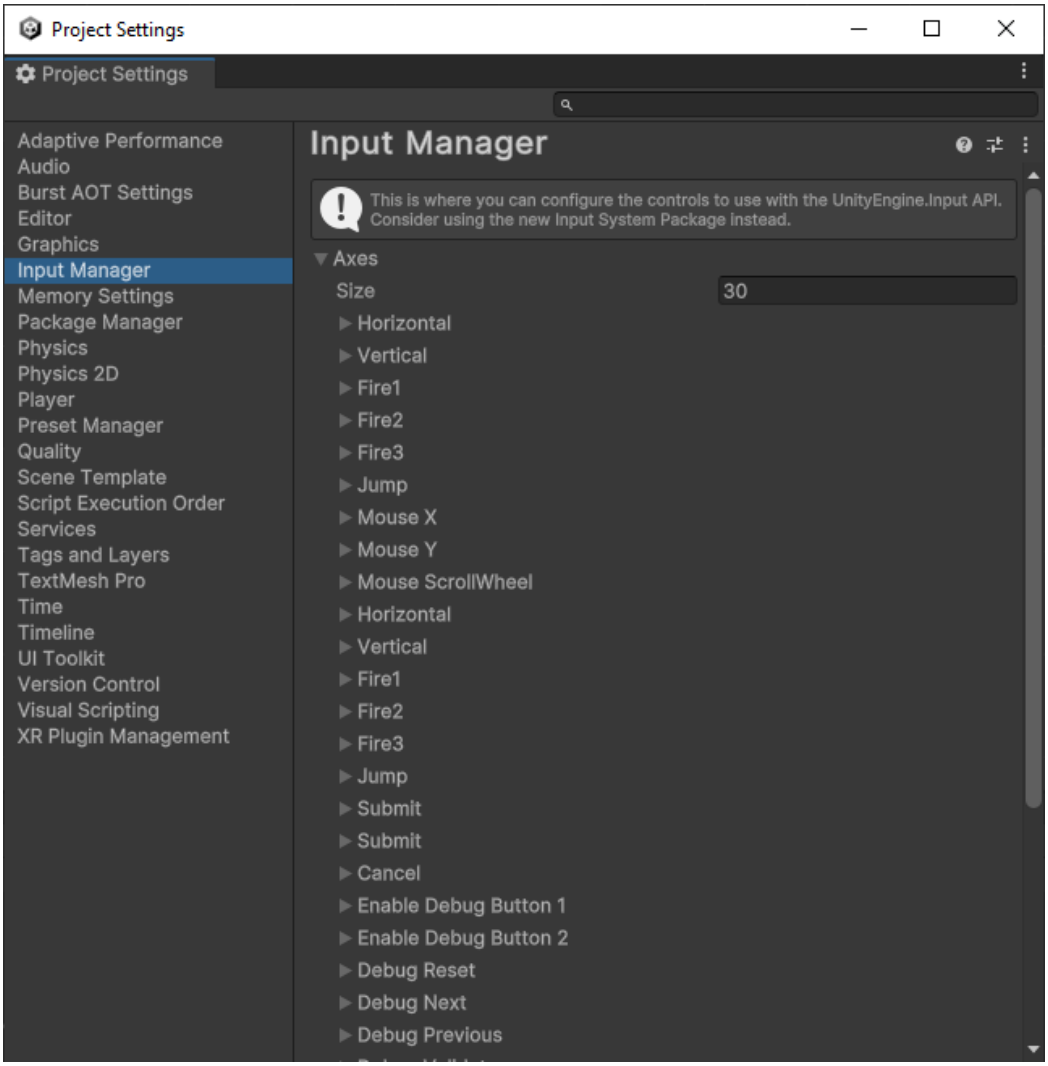
- **Player Settings**: Hộp thoại này giúp thiết lập một số tính năng người dùng khi thực thi như tên sản phẩm, bản quyền sản phẩm, phiên bản, Icon, chuột, hoặc độ phân giải màn hình…

Chức năng Player Settings là một phần con của Project Setting… sẽ được giới thiệu kỹ hơn trong menu Edit.

Sau khi đã điều chỉnh đầy đủ các tham số, chúng ta nhấn vào Build hoặc Build and Run để xuất bản dự án. Lưu ý là dự án chỉ được xuất bản thành công khi toàn bộ lỗi trong dự án đã được sửa.

Menu Edit: Menu này dùng để quản lý việc thay đổi các đối tượng và thành phần trên Scene. Ví dụ như copy, paste, lựa chọn đối tượng, tìm kiếm đối tượng, nhân bản đối tượng…

Chức năng **Project Settings**: Chức năng này cho phép thiết lập các thông số của dự án. Các thông số này chỉ được thiết lập tại dự án này, áp dụng cho tất cả các Scene nhưng không áp dụng cho dự án khác.



Hình 3.2. Chức năng Project Settings

Có rất nhiều các thông số mà Unity cho phép thiết lập trong dự án, phần sau đây giới thiệu một số thông số cơ bản:

- **Audio**: Thiết lập thông số về âm thanh như âm lượng tối đa, âm lượng tối thiểu, kiểu âm thanh.

- **Graphics**: Thiết lập thông số về đồ họa

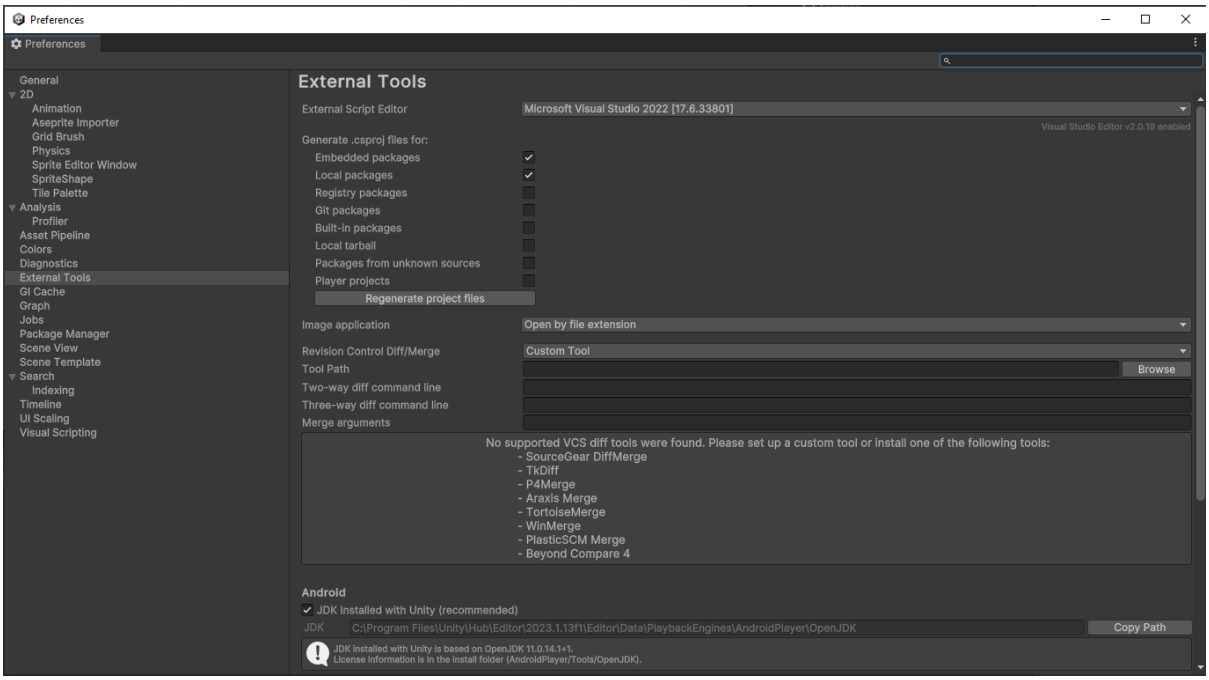
- Input manager: Các thông số về thiết bị đầu vào. Unity cho phép thiết lập các thông số đầu vào như quy định về nhấn phím ngang, dọc, phím khai hỏa hoặc các quy định về nhấn chuột.

- **Physics và Physics 2D**: Quy định một số tính chất vật lý của đối tượng như chất liệu mặc định, độ nẩy tối đa, hướng trọng lực, tốc độ quay…

- Time: Quy định về thời gian như thời gian kế tiếp, thời gian cập nhật frame…

Chức năng **Preferences**: Chức năng này cũng cho phép thiết lập một số thông số, nhưng sẽ áp dụng cho Unity, nghĩa là sẽ áp dụng cho các dự án khác.

- **2D**: Quy định một số thiết lập 2D như Animation, Sprite, Tile. Các phần nãy sẽ được giới thiệu kỹ hơn ở các chương sau.



Hình 3.3. Chức năng Preferences

- **Colors**: Quy định về màu sắc cho giao diện. Người dùng căn cứ vào các thông số để thiết lập màu sắc hợp lý.

- **External Tools**: Chức năng này cho phép thiết lập công cụ nào dùng để viết mã nguồn. Để thuận tiện trong viết mã lệnh với các gợi ý, trong phần External Script Editor, chúng ta cần chọn công cụ Microsoft Visual Studio (đánh dấu tick vào 2 lựa chọn là *Embedded packages* và *Local packages*) (Hình 3.3).

- Scene View: Quy định về giao diện hiển thị Scene

- Timeline: Quy định về cách thức biểu diễn thanh Timeline trong phần Animation (sẽ giới thiệu kỹ hơn ở phần sau).

Menu Assets: Menu này dùng để quản lý các assets, là các đối tượng được lưu trong thư mục assets. Các chức năng trong menu này bao gồm tạo mới assets (Create), nhập mới một asset… Khi tạo mới asset, đối tượng sẽ được lưu trong thư mục Assets của dự án, chúng ta có thể xem các đối tượng này bằng cách chọn menu Show in Explorer.

Menu GameObject: Đây là menu dùng để quản lý các đối tượng trong game như tạo mới đối tượng, nhân bản đối tượng, tạo quan hệ cha con cho đối tượng…

Menu Component: Menu này nhằm thêm các thành phần trong một đối tượng game. Để thực hiện được chức năng của menu này, cần chọn đối tượng game trên Hierarchy, sau đó chọn thành phần cần thêm.

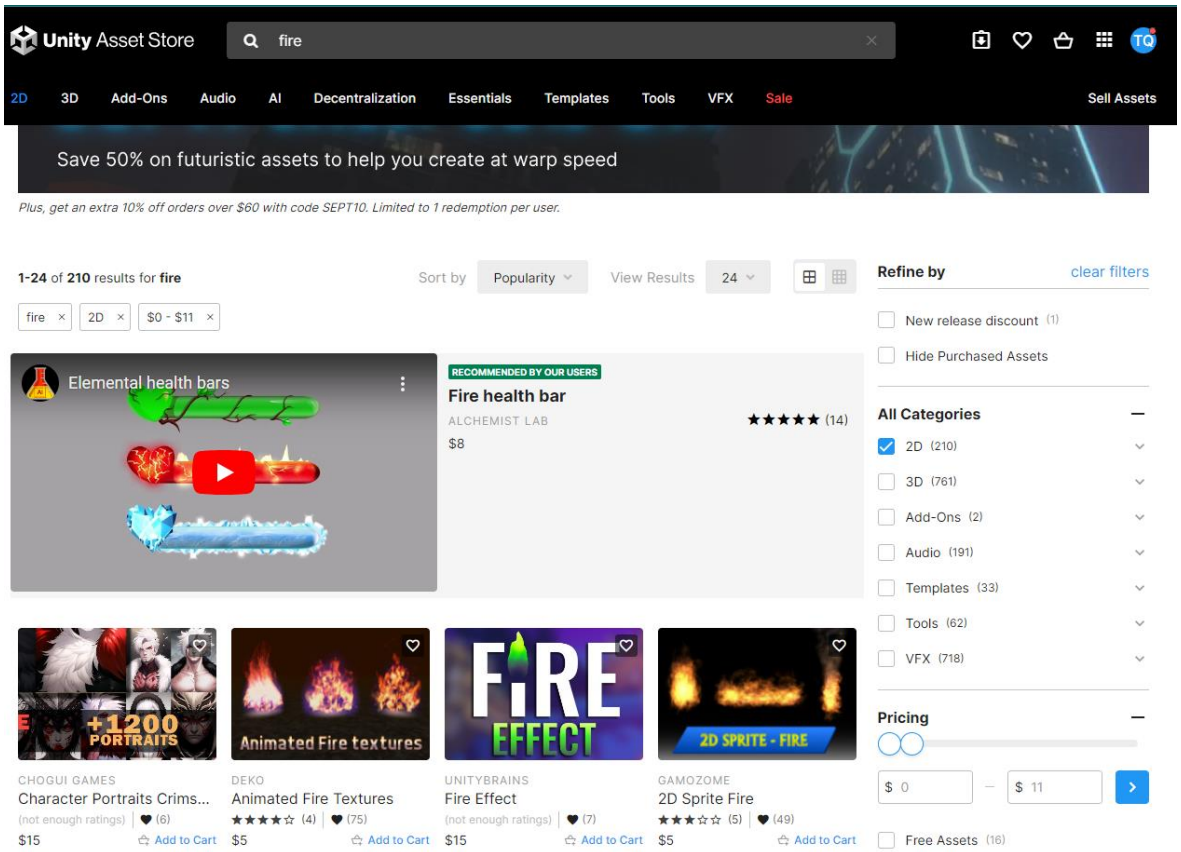
Menu Services: Quản lý dịch vụ của dự án, menu này gồm 2 chức năng là Explorer dùng để khám phá các dịch vụ do Unity cung cấp và General Settings dùng để thiết lập một số thông số cho dịch vụ.

Menu Jobs: Quản lý các tác vụ của dự án như đồng bộ hóa, hiển thị thời gian, …

Menu Window: Quản lý cửa sổ, menu này giúp người dùng ẩn và hiện các cửa số tương ứng. Người dùng có thể thực hiện bất, tắt các cửa sổ cần thiết như Animation, Rendering, Package Manager, Asset Store, …

Chức năng **Asset Store**: Asset Store là một nguồn tài nguyên được Unity cung cấp dưới dạng miễn phí hoặc có phí. Tài nguyên này được cộng đồng tạo ra, cho phép nhúng vào Unity để làm tài nguyên cho game. Cách thực hiện như sau:

Bước 1: Vào menu Window, chọn Asset Store, trang web Asset Store hiện ra:

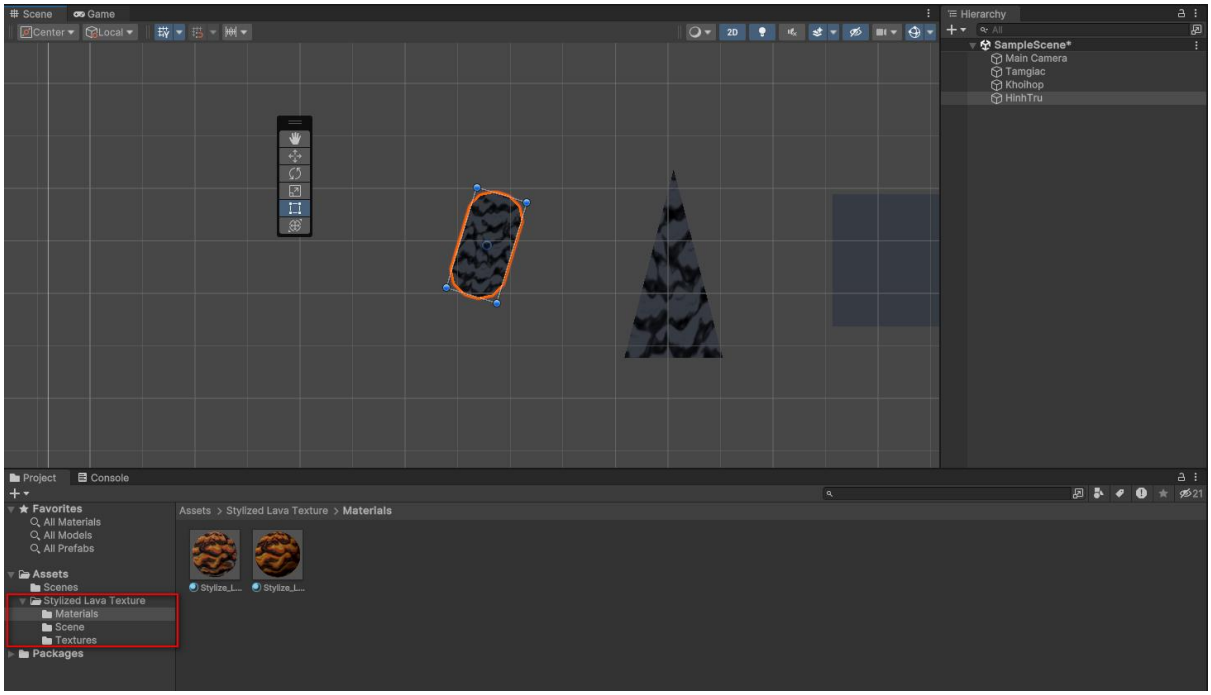


Hình 3.4. Màn hình trang web Unity Assets với tìm kiếm “fire” và một số tùy chọn

Bước 2: Gõ vào tên của các đối tượng cần tìm, ví dụ Stone (đá), Fire (lửa), Brick (gạch), Grass (cỏ)… chọn loại hình ở chức năng All Category (bên phải), sau đó chọn giá ở phần Pricing (lựa chọn miễn phí bằng cách kéo giá về 0).

Bước 3: Bấm chọn đối tượng, rồi chọn nút Add to My Asset, tiếp tục nhấn nút Open in Unity và làm theo một số bước để tải về và nhúng vào dự án. Khi hoàn tất, thành phần Asset này sẽ được đưa vào dự án, từ đây có thể dùng để đưa lên game (Hình 3.5).

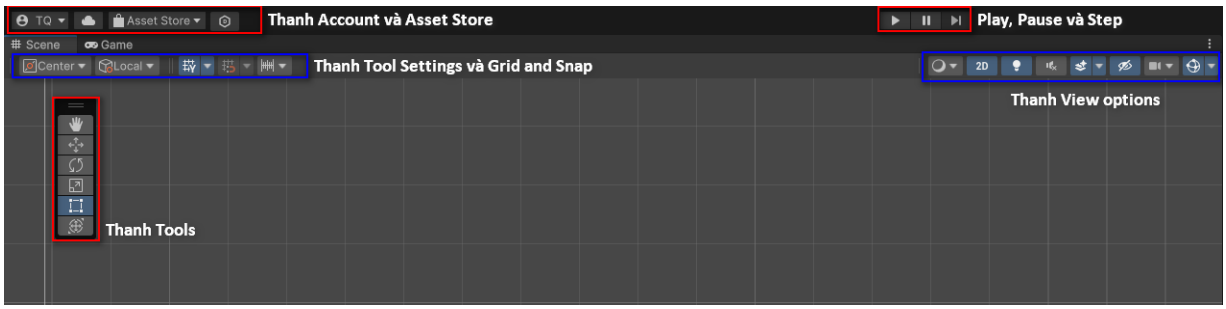
Menu Help: Dùng để hướng dẫn và trợ giúp trong Unity.



Hình 3.5. Đối tượng *Stylized Lava Texture* được tải về và nhúng vào dự án

2. Các chức năng cơ bản trên giao diện Scene

Trên giao diện Scene, Unity thiết lập sẵn 6 thanh chức năng bao gồm: Account và Asset Store; Play, Pause, and Step; Tool Settings; Grid and Snap; View options, và Tools (Hình 3.6).



Hình 3.6. Các thanh cơ bản trên Unity

- Thanh **Account và Asset Srore**: Thanh này hiển thị account hiện hành đang đăng nhập và đường dẫn tắt đến trang web Asset Store.

- Thanh **Play, Stop, và Step**: Là 3 nút dùng để điều khiển việc chạy, dừng game. Lưu ý là khi game đang được chạy, mọi thiết lập trên Scene, Hierarchy, hay Inspector đều chỉ là tạm thời.

- Thanh **Tools**: Thanh này có 6 chức năng:

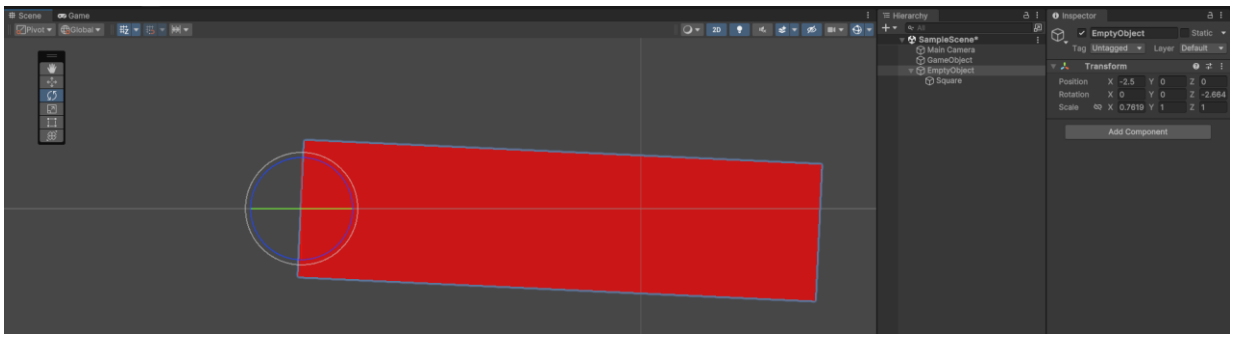
- ***Hand***: Cho phép chọn và di chuyển toàn màn hình bằng chuột trái hoặc phải.
- ***Move Tool***: Cho phép dịch chuyển vị trí. Khi chọn chức năng này, click vào đối tượng nào, ta có thể di chuyển đối tượng đó theo chiều X, Y hoặc X và Y (cả trục Z nếu chọn game 3D). Để ý là khi di chuyển đối tượng, giá trị Position X, Y trên phần Transform của Inspector cũng thay đổi theo.
- ***Rotate Tool***: Cho phép xoay đối tượng theo trục X hoặc Y (hoặc Z), điểm xoay mặc định là tâm của đối tượng. Khi xoay đối tượng, giá trị Rotation X, Y trên phần Transform của Inspector cũng thay đổi theo.

- ***Scale Tool***: Cho phép co dãn đối tượng theo trục X hoặc Y (hoặc Z), điểm co dãn mặc định là tâm của đối tượng. Khi co dãn đối tượng, giá trị Scale X, Y trên phần Transform của Inspector cũng thay đổi theo.
- **Rect Tool**: Cho phép điều khiển đối tượng bởi một hình chữ nhật bao quanh đối tượng. Chức năng này thường có tác dụng trong việc điều khiển các đội tượng có cấu trúc phức tạp.
- **Transform Tool**: Đây là chức năng tổng hợp hai trong một, cho phép vừa quay, vừa dịch chuyển đối tượng.

- Thanh **Tools Settings**: Chức năng này liên quan đến việc thiết lập đối tượng cha-con trong Unity. Nếu có một đối tượng nằm trong một đối tượng khác (quan hệ cha-con sẽ được giải thích rõ hơn ở phần sau) thì lúc này khi thực hiện các phép quay và co dãn thì dựa theo tọa độ nào để thực hiện. Nếu như chọn Pivot thì sẽ thực hiện các phép trên theo đối tượng cha, nếu là chọn Pivot sẽ thực hiện các phép trên theo đối tượng con.

Tương tự với Global và Local: Nếu chọn Global, tọa độ khi dịch chuyển sẽ được tính theo tọa độ của Scene, còn nếu chọn Local thì tọa độ được tính theo tọa độ của đối tượng.

***Thực hành**: Tạo 1 đối tượng Empty Object và một đối tượng là Square. Thiết lập Square có độ co dãn X là 5, Empty Object có vị trí X là -2.5, Square là con của Empty Object. Thực hiện các phép co dãn và quay; lần lượt chọn Center, Pivot, Global, Local để thấy kết quả.



Hình 3.7. Thực hành co dãn và xoay với Tool settings

- Thanh **Grid and Snap:** Cho phép thiết lập các thông số về lưới hiển thị như kích thước lưới, độ trong suốt hoặc phương pháp di chuyển đối tượng.

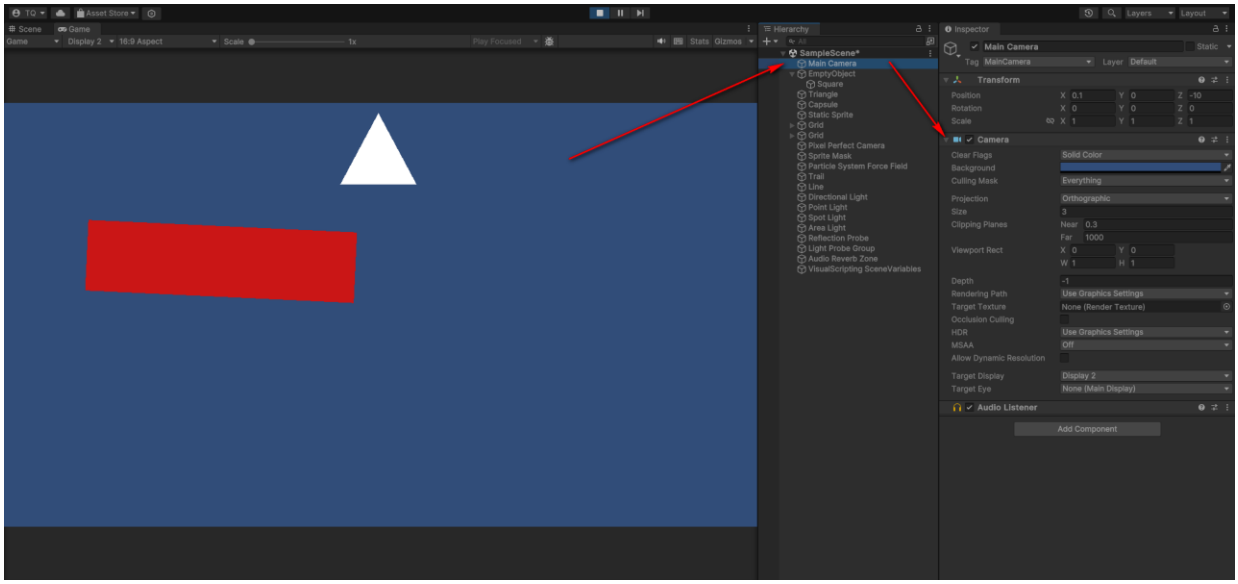
- Thanh **View Options**: Thanh này cho phép thiết lập một số chức năng liên quan đến hiển thị. Các chức năng chính bao gồm:

- **Draw mode**: Cho phép thiết lập các chế độ vẽ lên màn hinh như hiển thị dạng đổ bóng, dạng lưới hoặc một số quy định về ánh sáng.
- **2D View:** Cho phép bật tắt chế độ 2D sang 3D. Có nhiều trường hợp, chúng ta cần soi vật thể ở chế độ 3D, lúc này chức năng này sẽ phát huy tác dụng. Khi ở chế độ 3D, một công cụ gọi là Orientation hiện lên, cho phép soi vật thể theo chiều X, chiều Y hoặc chiều Z.
- **Scene Lighting**: Chế độ này cho phép sử dụng đèn mặc định của Scene, nếu tắt đi thì đèn được đưa vào Scene sẽ phát huy tác dụng.
- **Toggle audio:** Giúp quản lý âm thanh trong Scene.
- **Sky Box:** Giúp quản lý nền trời phía sau game, chức năng này thường có tác dụng trong môi trường Game 3D.
- **Toggle Scene object:** Thanh này giúp quản lý các đối tượng bị ẩn trên Scene.

- **Scene Camera:** Quản lý đối tượng camera như khung nhìn, phép chiếu… Phần camera sẽ được giải thích kỹ hơn ở phần sau.
- **Gizmos:** Chức năng này cho phép người dùng thiết lập các tính năng khác trong Unity như lựa chọn đối tượng, thiết lập mã nguồn, các hoạt cảnh, thiết lập khung nhìn… Đây là một chức năng có nhiều chức năng con, độc giả có thể tìm hiểu về từng chức năng thông qua các hoạt động thực hành sau này.

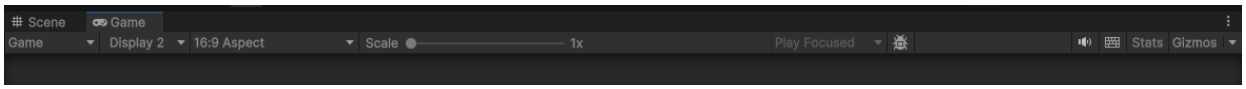
3. Các chức năng chính trên giao diện Game

Giao diện Game là giao diện được hiển thị khi nhấn nút Play. Giao diện này thường gắn liền với Camera chính của Game, khi thay đổi các tham số trên giao diện này cần chú ý đến các tham số trên Main Camera.



Hình 3.8. Giao diện Game

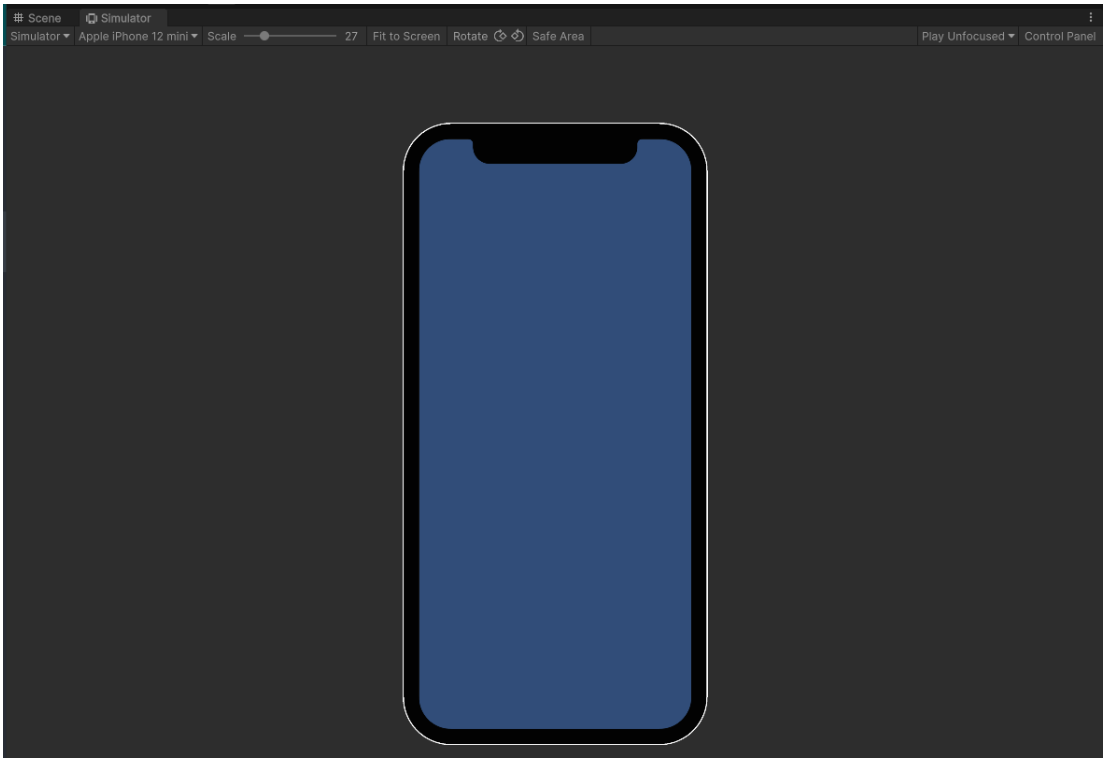
Trên giao diện Game, Unity cho phép chọn màn hình là Game hoặc Simulate. Nếu chọn Game, sẽ có các tham số như sau (Hình 3.9):



Hình 3.9. Các tham số với giao diện Game

- **Display**: Cho phép chọn chế độ hiển thị. Trong quá trình làm Game, có thể có nhiều Camera, mỗi camera sẽ có 1 chế độ hiển thị khác nhau. Chức năng này cho phép chọn chế độ hiển thị tương ứng với Camera.
- **Aspect**: Là cách thức hiển thị, có nhiều cách thức như Free Aspect, 16:9 Aspect, Full HD, QHD… Tùy vào mục đích của Game để chọn chế độ hiển thị phù hợp.
- **Scale**: Giúp phóng to hay thu nhỏ màn hình Game.
- **Speaker, Stats, Gizmos**: Giúp điều khiển âm thanh hoặc các trạng thái trên Game.

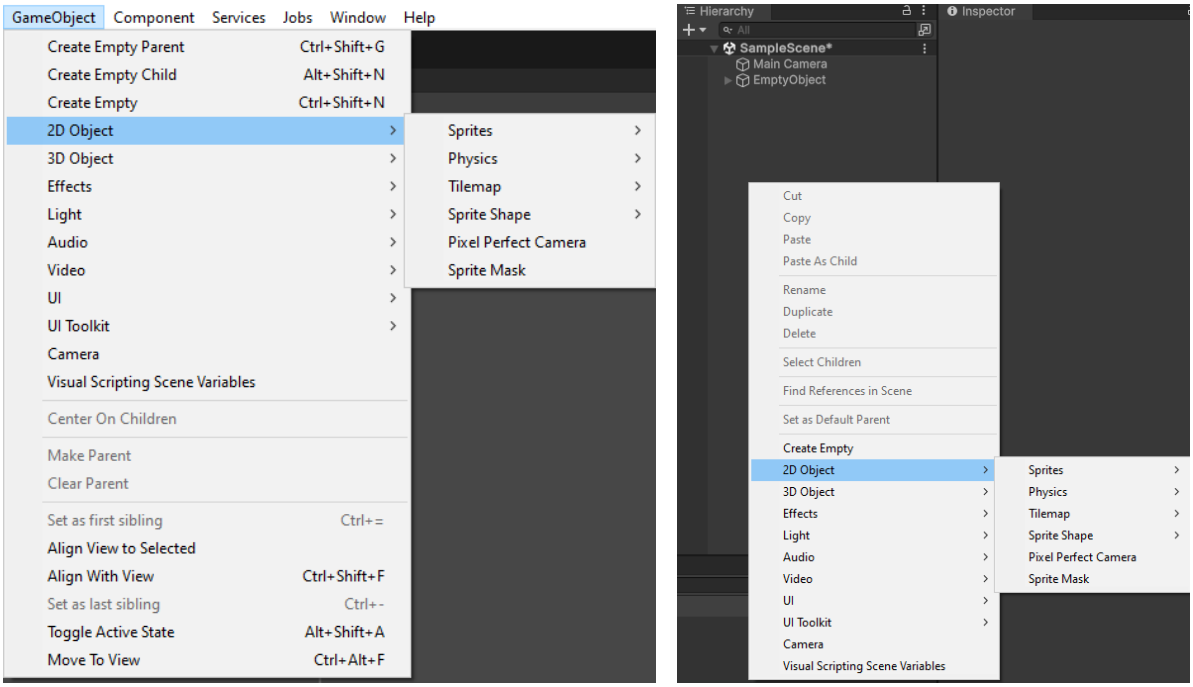
Nếu chọn Simulator, hệ thống giả lập màn hình điện thoại, người dùng có thể chọn các hang điện thoại tại mục Devices, chọn kích thước ở phần Scale hoặc thực hiện do dãn màn hình ở các phần tiếp theo (Hình 3.10)



Hình 3.10. Màn hình giao diện giả lập điện thoại

3. Đối tượng game

Đối tượng Game hay còn gọi là GameObject, là tất cả các thành phần được tạo ra trên một Scene. Các đối tượng Game này có thể nhìn thấy trong giao diện Scene và Hierarchy. Để tạo mới đối tượng Game, click phải lên Hierarchy hoặc vào GameObject rồi chọn các đối tượng Game tương ứng (Hình 3.11).



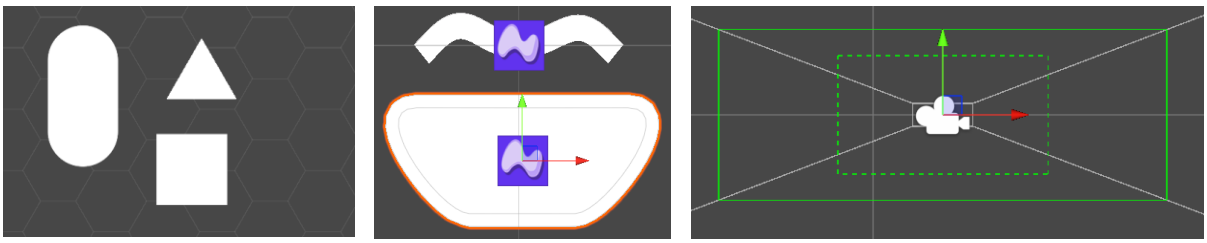
Hình 3.11. Tạo mới đối tượng Game bằng menu hoặc trên Hierarchy

Trong phạm vi phần này, tác giả chỉ giới thiệu tổng quan một số đối tượng game, chi tiết sẽ được giới thiệu kỹ hơn ở các phần sau.

- **Empty Object**: Đây là một đối tượng game rỗng, chỉ dùng để đánh dấu 1 vị trí hoặc điều khiển một đối tượng game khác. Mặc dù là đối tượng rỗng nhưng lại có vai trò quan trọng trong nhiều trường hợp.

- **2D Object**: Chứa các đối tượng game 2D như:

- ***Sprites***: Là các đối tượng hình ảnh 2 chiều (Tam giác, hình vuông, hình tròn, hình trụ tròn, …);
- ***Physics***: Là các đối tượng sprite nhưng được gán sẵn các thành phần vật lý nên có sẵn các tính chất vật lý.
- **Tilemap**: Là mạng lưới các đường kẻ để tạo bản đồ cho Game. Unity có sẵn các tilemap như hình chữ nhật, hình lục giác, hình thoi để người dùng lựa chọn.
- **Sprite Shape**: Unity thiết kế sẵn các sprite có hình dạng cho trước như Closed Shape, Open Shape.
- **Pixel Perfect Camera:** Là một dạng camera hỗ trợ đọc các hình ảnh có độ phân giải thấp.
- **Sprite Mask:** Là một công cụ dùng để thêm các hiệu ứng vào game bằng mặt nạ sprite.



Sprites và Tilemap | Sprite Shape | Pixel Perfect Camera

Hình 3.12. Một số hình ảnh của đối tượng Game 2D nguyên thủy

- **3D Object**: Là các đối tượng game 3D, các đối tượng này thường không được quan sát rõ trong môi trường 2D. Các đối tượng game 3D bao gồm: Hình cầu, hình hộp, hình trụ… Các đối tượng này sẽ được thấy rõ khi người dùng tạo dự án game 3D.

- **Effects**: Chứa các đối tượng dùng để tạo ra hiệu ứng cho game. Các đối tượng hiệu ứng bao gồm:

- ***Particle System***: Là hệ thống tạo ra các hạt, từ đó để tạo các hiệu ứng sinh động như khói, lửa, vụ nổ, pháo hoa…
- ***Particle System Force Field***: Là thành phần giúp tạo nên các ràng buộc lực lên hệ thống hạt.
- ***Trail:*** *Là thành phần giúp tạo ra một vệt sáng phía sau đối tượng.*
- ***Line:*** *Là thành phần giúp vẽ các đường trong Unity.*

Particle system | Line | Trail

Hình 3.13. Các hiệu ứng được tạo trong Unity (Nguồn: https://docs.unity3d.com)

- **Light**: Chứa các đối tượng tạo nên ánh sáng cho khung cảnh. Các đối tượng ánh sáng bao gồm:

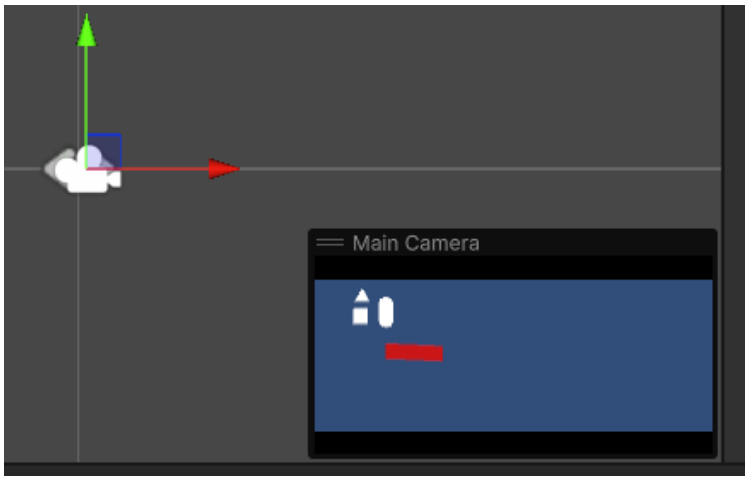
- ***Directional light***: Ánh sáng trực tiếp, là ánh sáng trực tiếp, không có hướng chiếu.
- ***Point light***: Ánh sáng điểm, chính là các điểm sáng tại một khu vực có giới hạn.
- ***Spot light***: Đây là dạng ánh sáng đèn pin, dùng để soi ở một khu vực nào đó.
- ***Area light***: Dạng ánh sáng vùng, chỉ có một vùng được chiếu sáng.
- ***Reflection probe*** và ***Light grobe group:*** Dạng ánh sáng phản chiếu, hắt qua một vật khác.

- **Audio**: Đối tượng này dùng để xử lý âm thanh, bao gồm Audio Source (nguồn âm thanh) và Audio Reverb Zone (âm thanh khu vực).

- **Video**: Giúp xử lý video trong game.

- **UI** và **UI Toolkit**: Đây là một tập các đối tượng dùng để giúp người dùng tương tác như hệ thống menu, ảnh nền, phần tùy chỉnh âm lượng… Chi tiết về phần này sẽ được đề cập ở Chương 8.

- Camera: Camera là một đối tượng quan trọng trong Unity. Camera giúp quan sát màn hình Game và hiển thị các đối tượng lên màn hình. Mỗi khi tạo một Scene mới, Unity sẽ tự động tạo ra một Camera mặc định. Người dùng có thể tùy chỉnh một số tham số trên phần Inspector như sau: Background (màu nền của Camera), Size (Kích thước Camera), Clipping Planes (Độ khuất của Camera), Target Display (Hiển thị ở màn hình nào). Camera có thể được điều khiển bằng mã lệnh.

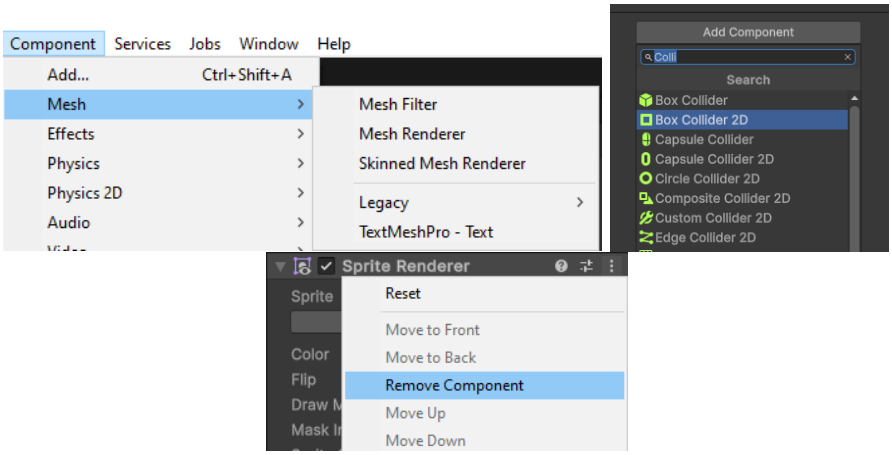


Hình 3.14. Camera với khung nhìn 2D

4. Các thành phần chính trong Unity

Trong Unity có rất nhiều Components (thành phần) dùng để gắn vào đối tượng Game và được nhìn thấy trên thanh Inspector. Các thành phần này sẽ quy định tính chất của đối tượng Game. Một số thành phần sẽ được gắn mặc định khi tạo mới đối tượng Game như Transform, Renderer… Một số thành phần khác sẽ tùy vào nhu cầu sử dụng mà có thể gắn thêm như Collider, Rigidbody, AudioSource, Script…

Để thêm thành phần mới, vào menu Component, chọn thành phần tương ứng, hoặc bấm vào nút Add Component trên thanh Inspector. Để xóa thành phần, bấm chuột vào dấu ba chấm ở góc trên, bên phải của thành phần, sau đó chọn Remove Component. Khi menu hiện ra, ta còn thấy chức năng Reset, chính là khôi phục lại các tham số mặc định của thành phần đó (Hình 3.15).



Hình 3.15. Hai cách thêm Component mới cho Game Object và cách xóa Component

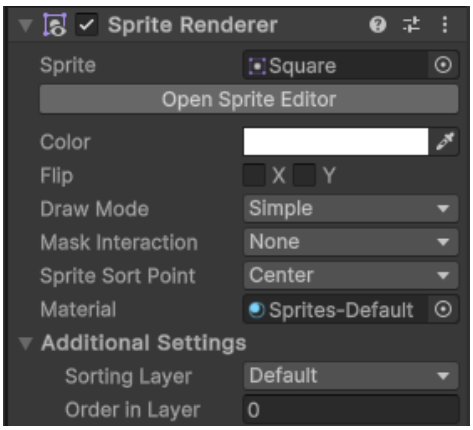
Phần sau đây sẽ giới thiệu một số thành phần cơ bản trong Unity, các thành phần khác sẽ được giới thiệu ở các chương tiếp theo.

- Thành phần **Transform**: Đây là thành phần mặc định của tất cả các đối tượng. Thành phần này bao gồm 3 đối số là Position quy định vị trí vật thể, Rotation quy định về góc quay của vật thể và Scale quy định về mức độ co dãn của vật thể. Thành phần này luôn mặc định và không thể xóa được.

Hình 3.16. Thành phần Transform

Để thay đổi các tham số, người dùng có thể dùng thanh công cụ để di chuyển đối tượng trên màn hình Scene hoặc nhập giá trị tương ứng vào ô hoặc bấm và rê chuột sang trái hoặc sang phải.

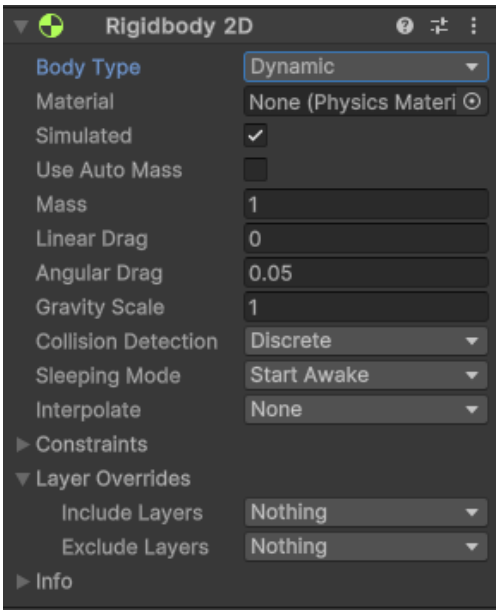
- Thành phần **Sprite Renderer**: Thành phần này giúp kết xuất Sprite và điều khiển cách nhân vật diễn ra trong một cảnh game. Sprite Renderer có các tham số như Sprite quy định về hình dạng đối tượng; Color quy định màu sắc của đối tượng; Flip quy định về độ lật của đối tượng; Draw mode là chế độ vẽ; Mask Interaction quy định cách thức giao thoa của vật thể và Material quy định về chất liệu của vật thể; Sorting Layer và Order Layer quy định về vị trí mà đối tượng này sẽ đứng trong Game.



Hình 3.17. Thành phần Sprite Renderer

Thành phần Sprite Renderer quan trọng nhất là phần Sprite. Nếu thêm vào một Sprite có sẵn thì mặc định sẽ là hình dạng cho trước. Nếu muốn thêm vào một nhân vật hoạt hình, người ta sẽ dùng Sprite Editor để cắt các khung hình phù hợp để đưa vào trong Sprite.

- Thành phần **Rigidbody**: Thành phần này quy định tính chất vật lý của đối tượng trong Game. Có hai loại là *Rigidbody* dùng cho đối tượng Game 3D và *Rigidbody2D* dành cho đối tượng Game 2D.



Hình 3.18. Thành phần Rigidbody2D

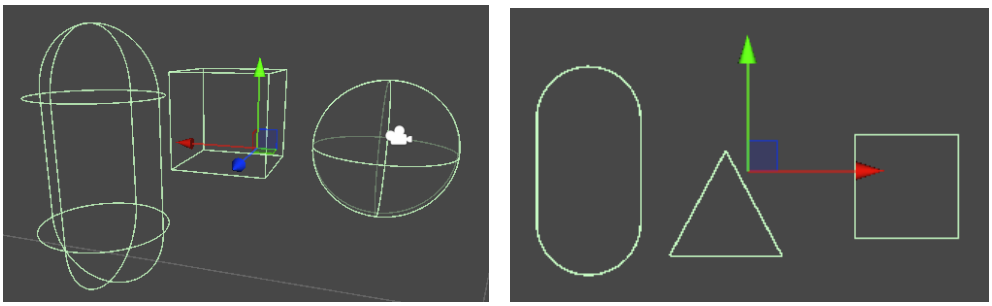
Thành phần Rigidbody có một số tham số cơ bản như trong Bảng 3.1:

Bảng 3.1 Các thuộc tính cơ bản trong thành phần Rigidbody2D

Thuộc tính	**Chức năng**
Body Type	Định nghĩa kiểu vật lý, nếu là Dynamic là dạng động, được thiết kế mặc định các tính chất, Kinematic là dạng động nhưng được thiết kế để điều khiển trong phần lập trình, và Static là dạng tĩnh, không điều khiển được.
Material	Đây là chất liệu vật lý cho đối tượng. Chất liệu vật lý sẽ có các tính chất như lực ma sát và độ nảy của đối tượng. Chất liệu vật lý có thể được tạo ra bằng cách trong Project, click phải chọn Physics Material
Simulated	Thuộc tính này nếu được đánh dấu, nó sẽ kết hợp với thuộc tính Dynamic của Body Type
Mass	Là khối lượng của vật thể. Nếu đánh dấu vào Use Auto Mass thì khối lượng sẽ được lấy mặc định, không phụ thuộc vào khối lượng người dùng nhập vào.
Drag	Lực cản tác động lên vị trí khi chuyển động (*Linear Drag*) hoặc khi quay (*Angular Drag*)
Gravity Scale	Định nghĩa góc mà Game Object bị ảnh hưởng bởi trọng lực
Collision Detection	Định nghĩa hình thức va chạm giữa các vật thể với hai loại là *Discrete* (va chạm rời rạc) *và Continuous* (liên tục)
Sleeping Mode	Thuộc tính này giúp cho GameObject rơi vào trạng thái “ngủ đông” khi ở chế độ nghỉ để tiết kiệm CPU.

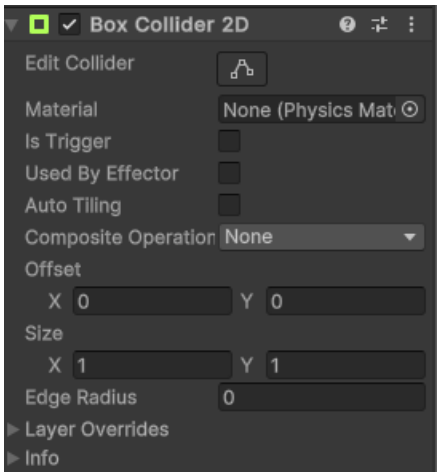
Constraints	Cố định đối tượng 2D khi nó chuyển động, có thể cố định theo vị trí (*Freeze Position*) hoặc cố định theo góc quay (*Freeze Rotation*).

- Thành phần **Collider**: Đây là thành phần giúp xác định sự va chạm của vật thể với một vật thể khác. Thành phần này là một đối tượng vô hình, bao xung quanh vật thể. Collider có nhiều loại, áp dụng cho các vật thể 2D hoặc 3D (Hình 3.19).



Hình 3.19. Một số loại Collider 3D (trái) và 2D (phải)

Một số thành phần *collider* được gắn sẵn với đối tượng game, một số thành phần khác thì người dùng phải tự thêm vào. Hình 3.20 biểu diễn 1 thành phần Box Collider 2D điển hình, một số tham số cơ bản được liệt kê trong Bảng 3.2.



Hình 3.20. Box Collider 2D

Bảng 3.2. Các thuộc tính cơ bản trong thành phần Box Collider 2D

Thuộc tính	Chức năng
Edit Collider	Đây là thuộc tính cho phép thay đổi vị trí các điểm trong Collider, khi click vào nút này, người dùng có thể thay đổi các điểm chấm bao quanh vật thể
Material	Đây là chất liệu vật lý cho đối tượng (xem Bảng 3.1)
Is Trigger	Thuộc tính này nếu được đánh dấu, nó sẽ kết hợp với thuộc tính Dynamic của Body Type
Mass	Là khối lượng của vật thể. Nếu đánh dấu vào Use Auto Mass thì khối lượng sẽ được lấy mặc định, không phụ thuộc vào khối lượng người dùng nhập vào.
Drag	Lực cản tác động lên vị trí khi chuyển động (*Linear Drag*) hoặc khi quay (*Angular Drag*)
Gravity Scale	Định nghĩa góc mà Game Object bị ảnh hưởng bởi trọng lực
Collision Detection	Định nghĩa hình thức va chạm giữa các vật thể với hai loại là *Discrete* (va chạm rời rạc) *và Continuous* (liên tục)

Sleeping Mode	Thuộc tính này giúp cho GameObject rơi vào trạng thái "ngủ đông" khi ở chế độ nghỉ để tiết kiệm CPU.
Constraints	Cố định đối tượng 2D khi nó chuyển động, có thể cố định theo vị trí (*Freeze Position*) hoặc cố định theo góc quay (*Freeze Rotation*).

- Thành phần **Audio Source**: Thành phần Audio Source giúp quản lý âm thanh trong Unity. Thành phần này có thể là một Game Object (nếu nó được thêm vào Hierarchy), hoặc là một Component (nếu nó được thêm vào một Game Object). Để thêm đối tượng Audio Source, click phải lên Hierarchy, chọn Audio, chọn Audio Source. Hoặc vào một Game Object, chọn Add Component, chọn Audio Source. Hình 3.21 minh họa Audio Source, Bảng 3.3. minh họa một số thuộc tính cơ bản.

Hình 3.21. Thành phần Audio Source

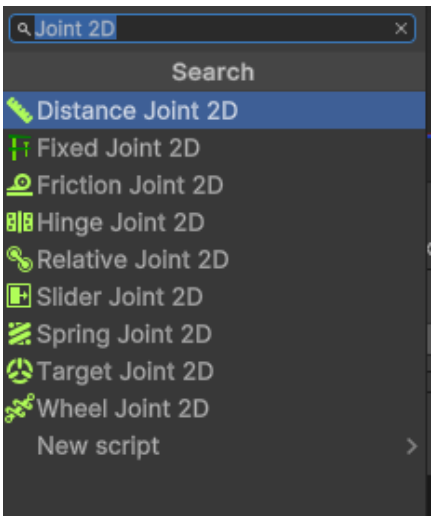
Lưu ý: Thành phần này quan trọng nhất là tập tin âm thanh, cần tải về tập tin âm thanh trước và để nó trong Project, sau đó kéo vào thuộc tính Audio Clip. Nếu muốn thiết lập nhạc nền, cần thiết lập tập tin nhạc tại Audio Clip, để chế độ lặp bằng cách tick chọn Loop.

Bảng 3.3. Một số thuộc tính chính của Audio Source

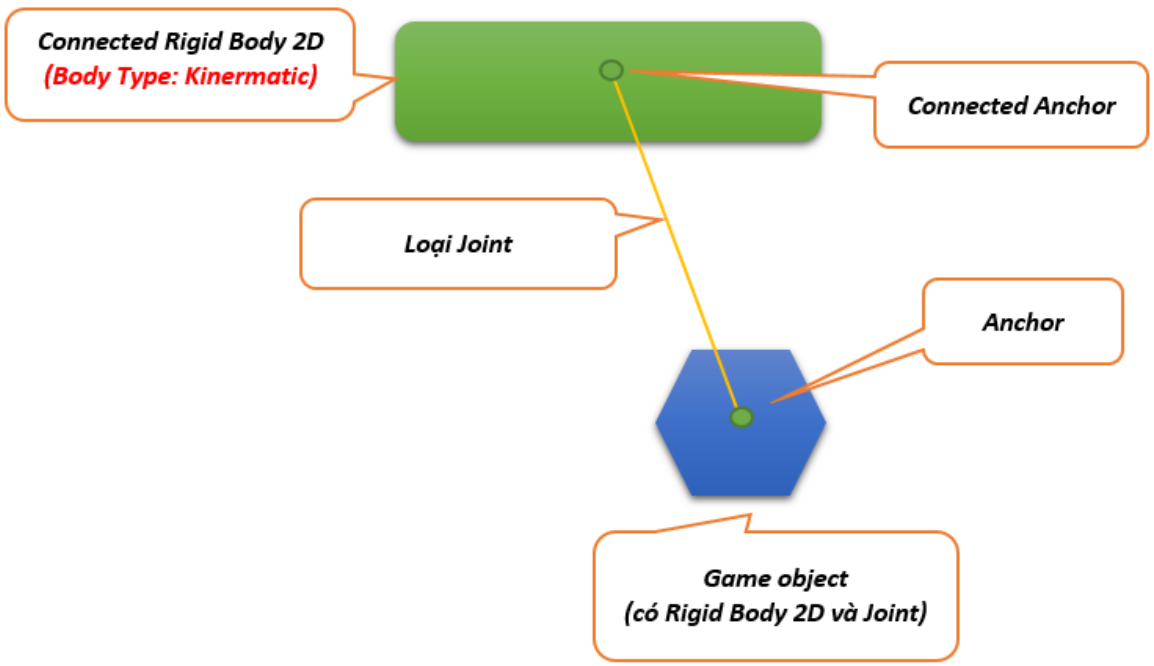
Thuộc tính	Chức năng
Audio Clip	Tham chiếu tới tập tin âm thanh cần chạy
Output	Mặc định âm thanh sẽ được xuất trực tiếp lên Audio Listener của Scene
Mute	Nếu được chọn, âm thanh sẽ bị tắt
Play On Awake	Nếu được chọn, âm thanh sẽ tự động chơi ngay khi Scene được khởi chạy
Loop	Lặp lại bài hát hoặc âm thanh đã phát
Volume	Tinh chỉnh âm lượng của âm thanh
Pitch	Tinh chỉnh cao độ của âm nhạc, giúp nhạc chạy nhanh hoặc chậm
3D Sound Settings	Tùy chỉnh âm thanh trong không gian, giúp nghe được âm thanh khi đến gần vật thể hoặc không nghe âm thanh khi ở xa vật thể
Rolloff Mode	Xác định các thể loại biến mất của âm thanh

5. Thành phần kết nối (Joint 2D) trong Unity

Thành phần Joint giúp gắn kết vật này vào vật khác. Cứ tưởng tượng, ta có 1 quả cầu đung đưa trên một sợi dây xích, một bánh xe lăn trên đường hoặc một dây chun có thể con dãn để kéo một vật hoặc 1 cánh cửa gắn với một bản lề. Tất cả những điều đó được thể hiện thông qua Joint. Unity hỗ trợ nhiều thành phần kết nối như Distance Joint 2D, Fixed Joint 2D, Friction Joint 2D, Hinge Joint 2D, Relative Joint 2D, Slider Joint 2D, Spring Joint 2D, Target Joint 2D và Wheel Joint 2D (Hình 3.22). Các thành phần Joint 2D bắt buộc đi kèm với thành phần Rigidbody 2D, nên khi thêm mới thành phần này vào thì hệ thống tự động thêm Rigidbody 2D. Để thêm thành phần Joint vào Game Object, chọn Add Component, gõ Joint 2D, chọn loại kết nối tương ứng. Các thuộc tính chung của thành phần Join được thể hiện như trong Bảng 3.4, ngoài ra mỗi thành phần sẽ có các thuộc tính đặc trưng riêng, sẽ được giới thiệu sau. Hình 2.23 minh họa một số thuộc tính của Joint 2D.



Hình 3.22. Các thành phần kết nối Joint 2D

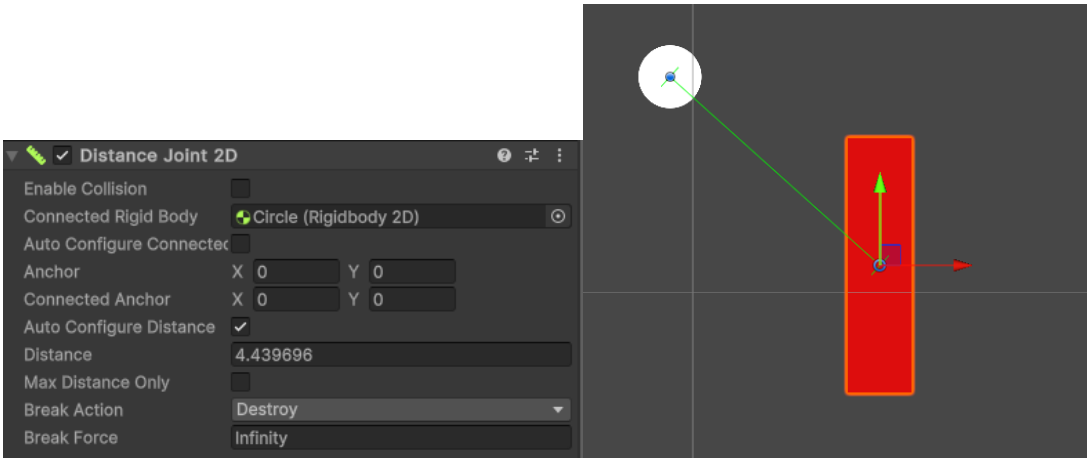


Hình 3.23. Một số thuộc tính chung trong Joint 2D

Bảng 3.4. Các thuộc tính chung của Joint 2D

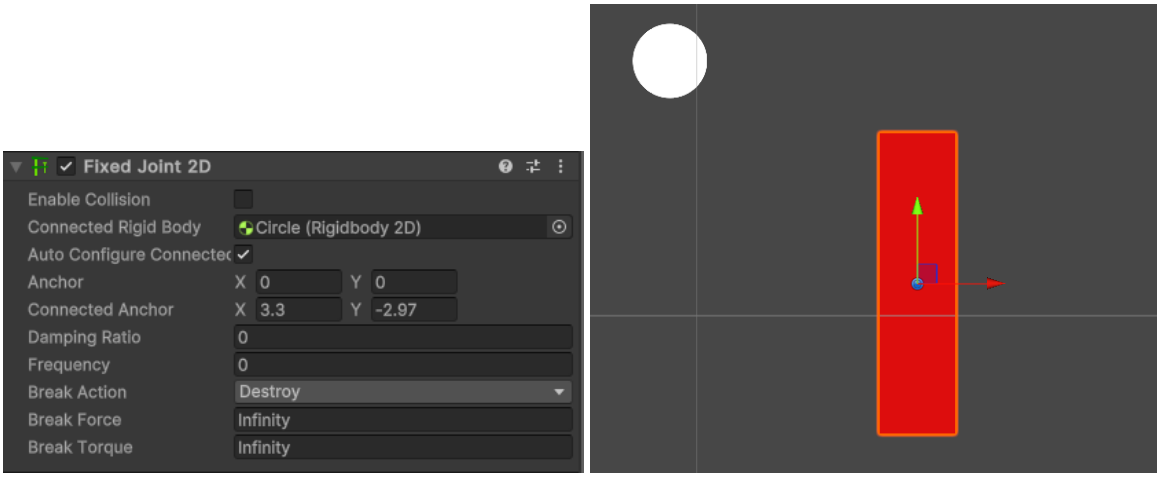
Thuộc tính	Chức năng
Enable Collision	Thuộc tính này nếu được kích hoạt, sẽ cho phép xử lý va chạm giữa 2 vật
Connected Rigid Body	Đây chính là đối tượng sẽ dùng để kết nối tới, lưu ý đối tượng này phải chứa Rigid Body 2D. Và trong thuộc tính này, phải thiết lập Body Type về Kinematic.
Auto Configure Connected Anchor	Tự tìm điểm neo của đối tượng được kết nối tới. Nếu được bật, bạn không cần nhập tọa độ trong phần *Connected Anchor.*
Anchor	Là tọa độ dùng để neo, mặc định là 0,0 nghĩa là ở tâm của đối tượng
Connected Anchor	Là tọa độ neo trên vật thể kết nối tới, mặc định là 0,0 nghĩa là ở tâm đối tượng
Break Force	Lực để bẻ gãy kết nối. Nếu để Infinity là không bao giờ bị bẻ gãy
Break Action	Hành động để bẽ gãy nối kết, mặc định là Destroy

- ***Thành phần Distance Joint 2D:*** Thành phần này giúp gắn kết 2 vật theo một khoảng cách nhất định. Thuộc tính Distance quy định khoảng cách giữa 2 vật thể, thuộc tính Max Distance Only quy định khoảng cách tối đa của hai vật thể đó (Hình 3.24)



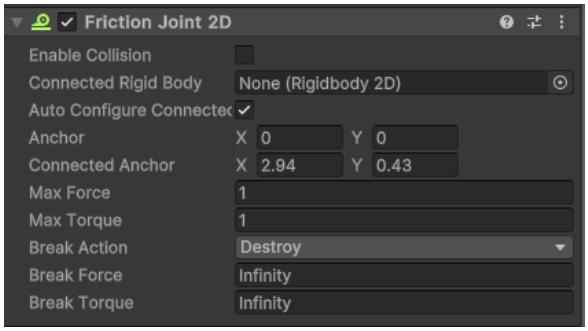
Hình 3.24. Minh họa thành phần Distance Joint 2D

- ***Thành phần Fixed Joint 2D*** giúp gắn một vật vào một vật khác, thành phần này có thêm thuộc tính Damping Ratio là mức độ giao động nhanh hoặc chậm và Frequency là tần số dao động



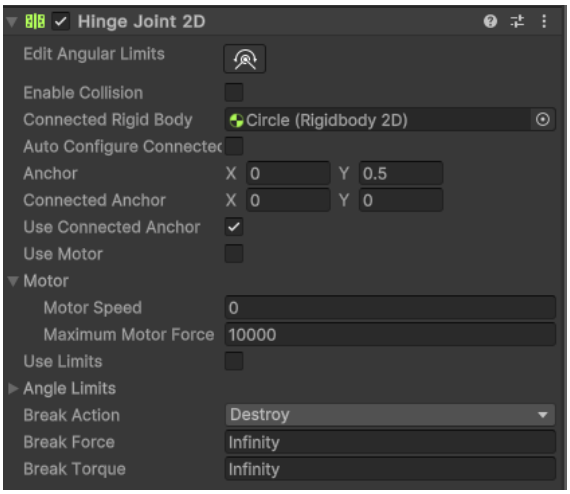
Hình 3.25. Minh họa thành phần Fixed Joint 2D

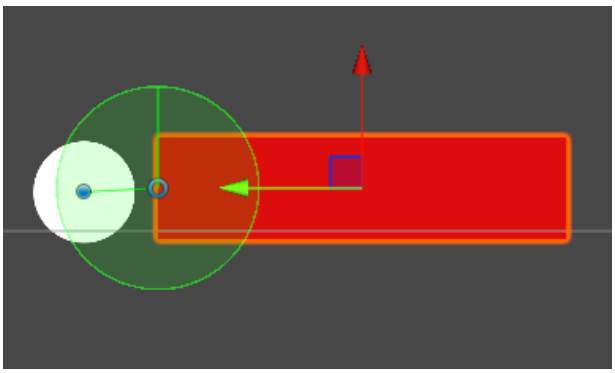
- Thành phần Friction Joint 2D cho phép mô phỏng hiệu ứng ma sát giữa hai vật thể, kiểm soát sự trượt của các đối tượng trong không gian 2D, giúp chúng tương tác với nhau theo cách thực tế hơn. Loại kết nối này thường dùng để minh họa quá trìn một vật thể bị ảnh hưởng mởi ma sát không khí hoặc ma sát với một vật thể khác.



Hình 3.26. Thành phần Friction Joint 2D

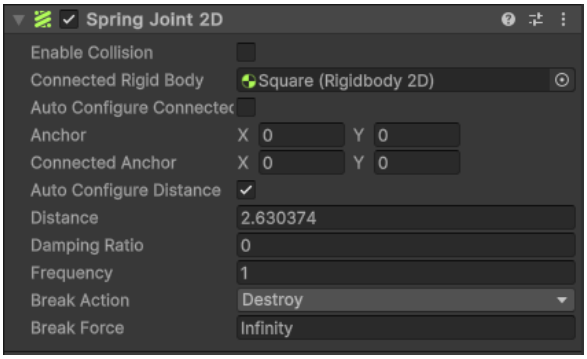
- Thành phần Hinge Joint 2D cho phép mô phỏng gắn bản lề giữa 2 đối tượng. Khi hai đối tượng gắn với nhau bằng Hinge Joint 2D, vật thể có thể rơi xuống nhưng sẽ bị xoay tròn theo điểm đã gắn, việc này giống như hai vật gắn vào nhau bởi bản lề. Hinge Joint 2D có một số thuộc tính quan trọng như Motor là thiết lập động cơ cho phép quay, Angle là góc quay tối đa của vật (Hình 3.27).

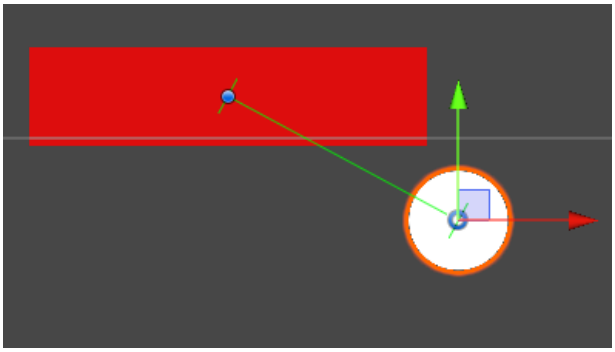




Hình 3.27. Minh họa thành phần Hinge Joint 2D

- Thành phần Spring Joint 2D cũng giống như Hinge Joint 2D, Spring Joint 2D cho phép gắn vật này vào vật khác nhưng bằng dây cao su, khi đó sức nặng của vật sẽ làm cho vật rơi xuống nhưn sẽ bị níu lại bằng một dây cao su vô hình (Hình 3.28). Thành phần này có thêm một số thuộc tính liên quan như Distance là khoảng cách lò xo, Damping Ratio là tỉ lệ co dãn của lò xo, Frequency là tần số giao động của lò xo.



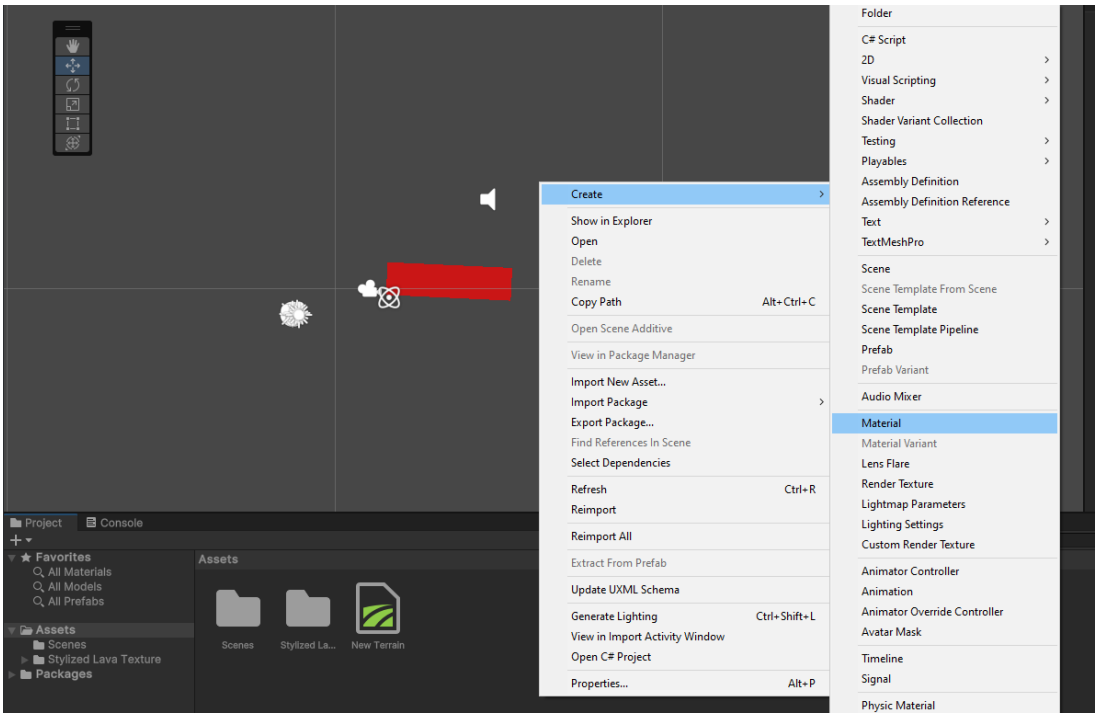


Hình 3.28. Minh họa thành phần Hinge Joint 2D

Ngoài những thành phần kết nối được nêu trên đây, Unity còn có rất nhiều thành phần kết nối khác như Target Joint 2D và Wheel Joint 2D… Các thành phần kết nối này cũng có tính chất tương tự như các thành phần đã nêu.

6. Chất liệu trong Game

Trong Unity, chất liệu được dùng để tạo nên bề mặt cho vật thể. Chất liệu có hai loại là chất liệu thường dùng (Material) và chất liệu vật lý (Physic Material). Để tạo chất liệu, click phải lên thư mục trong Project, chọn Create, chọn Material (Hình 3.29). Thực hiện tương tự với Physic Material và Prefab.



Hình 3.29. Tạo mới Material

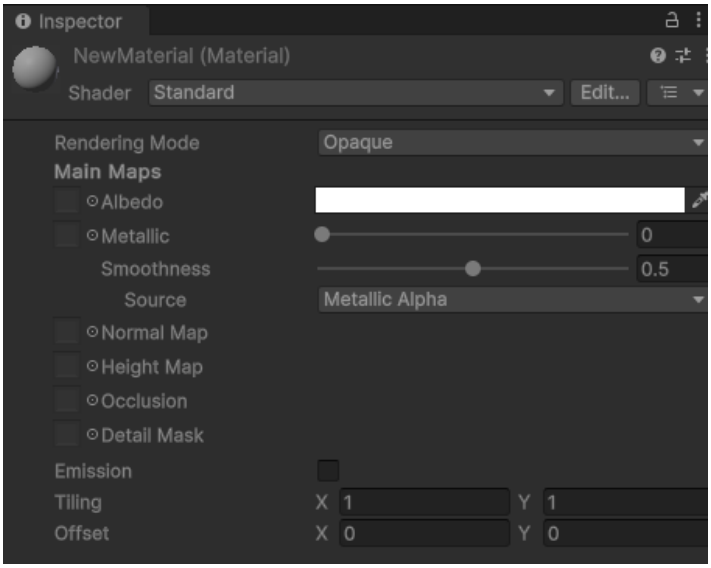
Lưu ý: Để dễ dàng quản lý Project, người ta thường tạo ra các thư mục trong Assets của Project và tạo các đối tượng trong đó. Ví dụ thư mục Scenes sẽ lưu các Scene, thư mục Materials sẽ lưu các chất liệu, thư mục Scripts sẽ lưu các mã nguồn, thư mục Prefabs sẽ lưu các Prefab.

- **Chất liệu Material**: Là dạng chất liệu giúp tạo hình cho đối tượng game. Có thể tạo hình cho đối tượng game bằng màu sắc thông thường hoặc màu ánh kim loại hoặc từ một hình nào đó. Material có các thuộc tính như trong Bảng 3.3.

Bảng 3.3. Các thuộc tính của Material

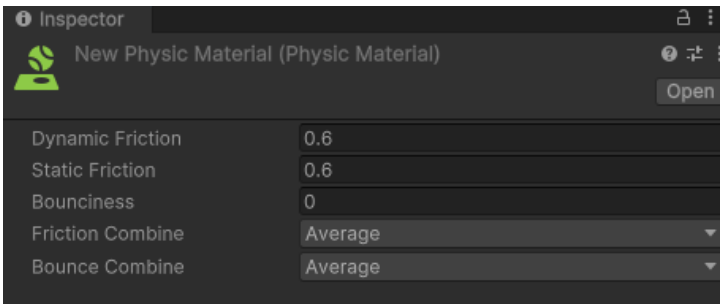
Thuộc tính	Chức năng
Shader	Là các dạng hình cụ thể của Material. Unity cho phép thiết lập hình dạng theo các dạng cụ thể nhất định. Người dùng có thể thiết lập các tham số mặc định bằng cách chọn Standard
Render Mode	Chế độ hiển thị, có thể thay đổi chế độ này bằng cách viết lệnh. Unity có các chế độ như Opaque: hiển thị mặc định, không trong suốt; Cutout: hiển thị với chế độ trong suốt với texture; Transparent: hoàn toàn trong suốt và Fade: là chế độ đổ bóng.
Albedo	Thuộc tính cho phép chọn màu sắc hoặc hình ảnh để hiển thị lên vật thể.

Metalic	Thuộc tính này cho phép thể hiện ánh kim lên vật thể, nghĩa là cho phép hiển thị vật thể theo dạng kim loại
Smoothness	Thuộc tính cho phép thiết lập mức độ làm trơn của vật thể
Normal Map	Cho phép thêm vào phần gồ ghề của một bức ảnh
Tiling và Obset	Tham số thể hiện mật độ của bức ảnh và khoảng cách phù hợp giữa các ảnh



Hình 3.30. Các thuộc tính chính của Material

- **Chất liệu Physic Material**: Là dạng chất liệu vật lý dùng để áp tính chất vật lý lên đối tượng. Chất liệu này được gắn vào trong thành phần RigidBody của các đối tượng Game. Physic Material có các thuộc tính như trong Bảng 3.4.



Hình 3.31. Các thuộc tính của Physic Material

Bảng 3.4. Các thuộc tính của Physic Material

Thuộc tính	Chức năng
Dynamic Friction	Ma sát động, đây là dạng ma sát khi vật thể chuyển động lên một vật thể khác, hệ số ma sát động sẽ từ 0 đến 1.
Static Friction	Ma sát tĩnh, đây là dạng ma sát khi vật thể va chạm với một vật thể khác, hệ số ma sát động sẽ từ 0 đến 1.
Bounciness	Thuộc tính này cho phép thiết lập độ nẩy của vật thể khi va chạm, độ nảy có giá trị từ 0 đến 1
Friction Combine	Thuộc tính này cho phép tính toán lực ma sát giữa hai vật thể, có các phương pháp tính như Average (trung bình), Minimum (Tối thiểu), Multiply (Nhân lên), Maximum (Tối đa)
Friction Combine	Thuộc tính này cho phép tính toán độ nẩy giữa hai vật thể khi chạm nhau, các phương pháp tính cũng như của lực ma sát.

7. Thiết lập Prefab

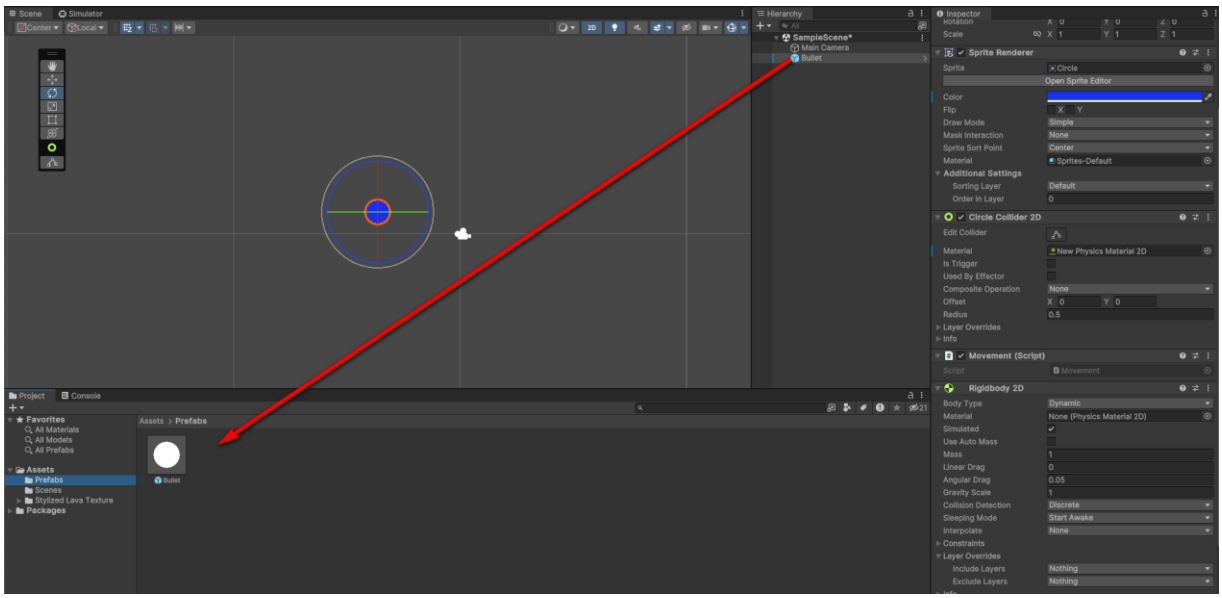
Trong lập trình Game, Prefab là một khái niệm quan trọng dùng để thiết lập đối tượng sau này có thể dùng lại nhiều lần. Ví dụ: Tạo 1 viên đạn để sau này bắn nhiều lần, tạo 1 vật cản để nó xuất hiện ngẫu nhiên, tạo 1 bông hoa để nó rơi ngẫu nhiêu… Cách thực hiện như sau (Hình 3.32):

Bước 1: Trên Hierarchy, tạo 1 đối tượng cần thực hiện, tiến hành thêm các thành phần cần thiết, viết các script cần thiết.

Bước 2: Nhấp chọn, kéo đối tượng vào thư mục chứa ở Project.

Bước 3: Xóa đối tượng trên Hierarchy. Bây giờ, đối tượng đã là một Prefab, nằm trong Project.

Bước 4: Khi cần dùng, có thể viết code để gọi đối tượng hoặc kéo đối tượng lên Scene. Có thể kéo nhiều lần, mỗi lần là một đối tượng game. Khi thay đổi các tính chất trong đối tượng ở Project, các đối tượng ánh xạ trên Scene sẽ thay đổi theo.



Hình 3.32. Thiết lập Prefab

8. Kết chương

Chương này giúp độc giả làm quen với các thành phần chính trong Unity như cách thiết lập cho người chơi, hệ thống menu, các thanh công cụ trên giao diện chính, các đối tượng 2D trong game, các thành phần và mối quan hệ giữa chúng. Ngoài ra, chương này cũng giới thiệu các thành phần kết nối Joint và các chất liệu để thực hiện hiệu ứng trong game.

Bài tập

1. Trong Project Settings hãy giải thích các tham số trong phần Horizontal và Vertical. Nếu muốn vật rơi nằm ngang thì cần điều chỉnh tham số gì? Giải thích chức năng External Tools.

2. Hãy thực hành nhúng một mô hình 2D từ Asset Store về dự án Unity.

3. Tạo 1 đối tượng Empty Object và một đối tượng là Square. Thiết lập Square có độ co dãn X là 5, Empty Object có vị trí X là -2.5, Square là con của Empty Object. Thực hiện các phép co dãn và quay; lần lượt chọn Center, Pivot, Global, Local để thấy kết quả.

4. Hãy tạo các đối tượng 2D Object, thêm màu, đặt tên cho đối tượng. Nhấn Play để thấy kết quả.

5. Giải thích các tham số chính trong Rigidbody2D.

6. Giải thích các tham số chính của Box Collider 2D.

7. Audio Source dùng để làm gì? Thực hành tạo nhạc nền cho game bằng Audio Source

8. Trong Unity, hãy thực hành các thành phần kết nối như Distance Joint 2D, Fixed Joint 2D, Friction Joint 2D, Hinge Joint 2D, và Spring Joint 2D. Giải thích vai trò của thuộc tính Connected Regid Body, Anchor và Connected Anchor trong các thành phần kết nối này.

9. Trong Unity, chất liệu là gì? Hãy nêu sự khác nhau giữa Material và Physic Material.

10. Prefab là gì? Cách tạo một Prefab.

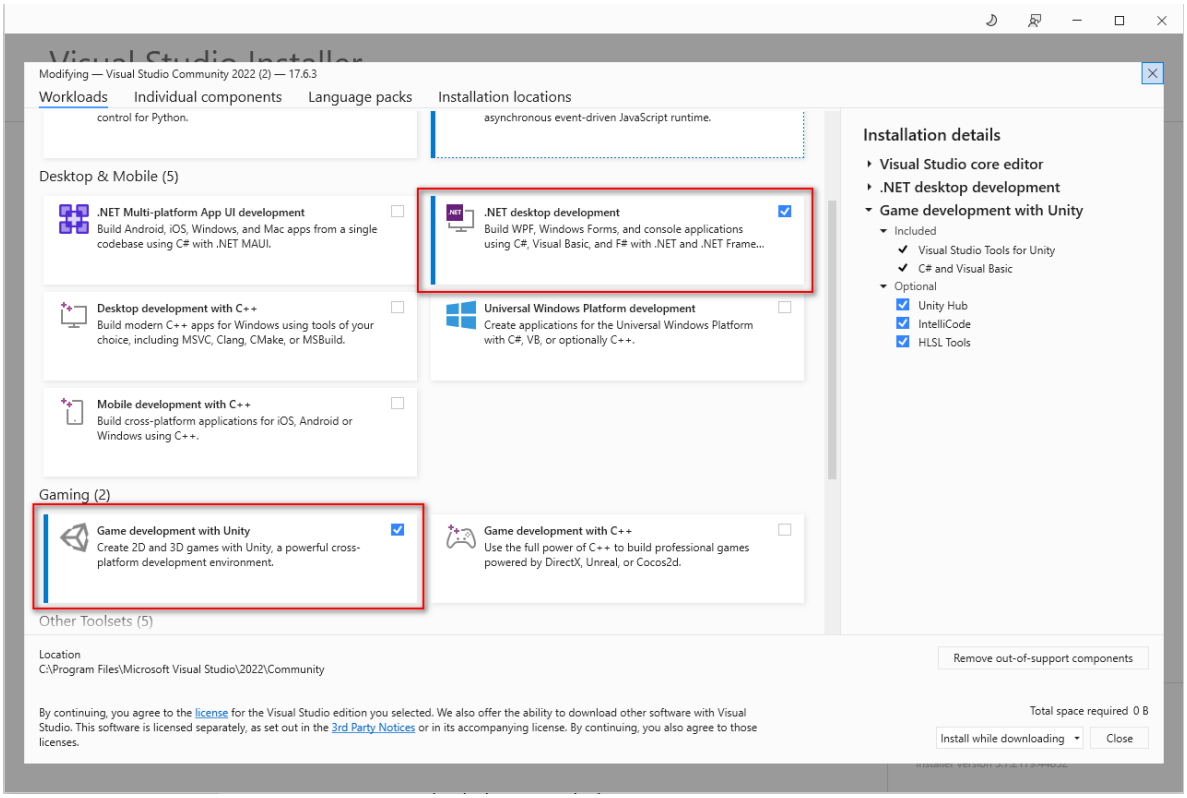
Chương 4. Kỹ thuật viết lệnh C# trong Unity

Một trong những kỹ thuật quan trọng trong lập trình Game đó chính là viết lệnh. Unity hỗ trợ việc viết lệnh bằng nhiều ngôn ngữ, nhưng phổ biến và được nhiều người dùng vẫn là C#. Chương này sẽ giới thiệu cách tạo lệnh và viết lệnh trong Unity bằng C# dựa trên bộ Visual Studio.

1. Tạo lệnh trong Unity

Unity hỗ trợ 3 ngôn ngữ lập trình là C#, Java Script, và Boo (một ngôn ngữ tựa Python). Trong ba ngôn ngữ này thì C# vẫn là ngôn ngữ được cộng đồng sử dụng nhiều nhất vì dễ sử dụng và có thể tích hợp vào trong bộ Visual Studio và Visual Studio Code. Giáo trình này chỉ hướng dẫn bằng ngôn ngữ lập trình C# trên bộ Visual Studio 2022 và trên hệ điều hành Window. Các thao tác cài đặt với Visual Studio Code, hoặc cài đặt trên hệ điều hành khác Window, độc giả có thể tìm kiếm trên Internet.

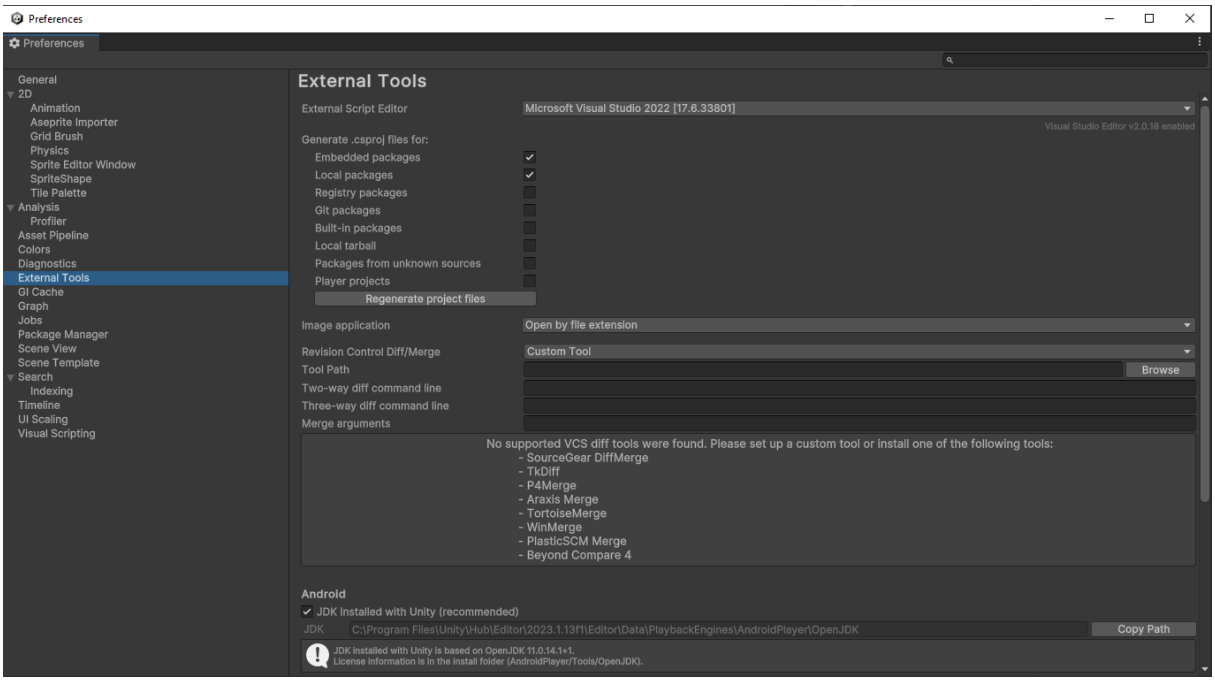
Để cài đặt, cần tải bộ Visual Studio Installer từ trang web của Microsoft, sau đó cài đặt các gói liên quan. Để Visual Studio hỗ trợ tốt viết mã lệnh và gợi ý mã lệnh với C# trong Unity, cần chọn 2 mục là **.NET desktop development** và **Game development with Unity** (Hình 4.1).



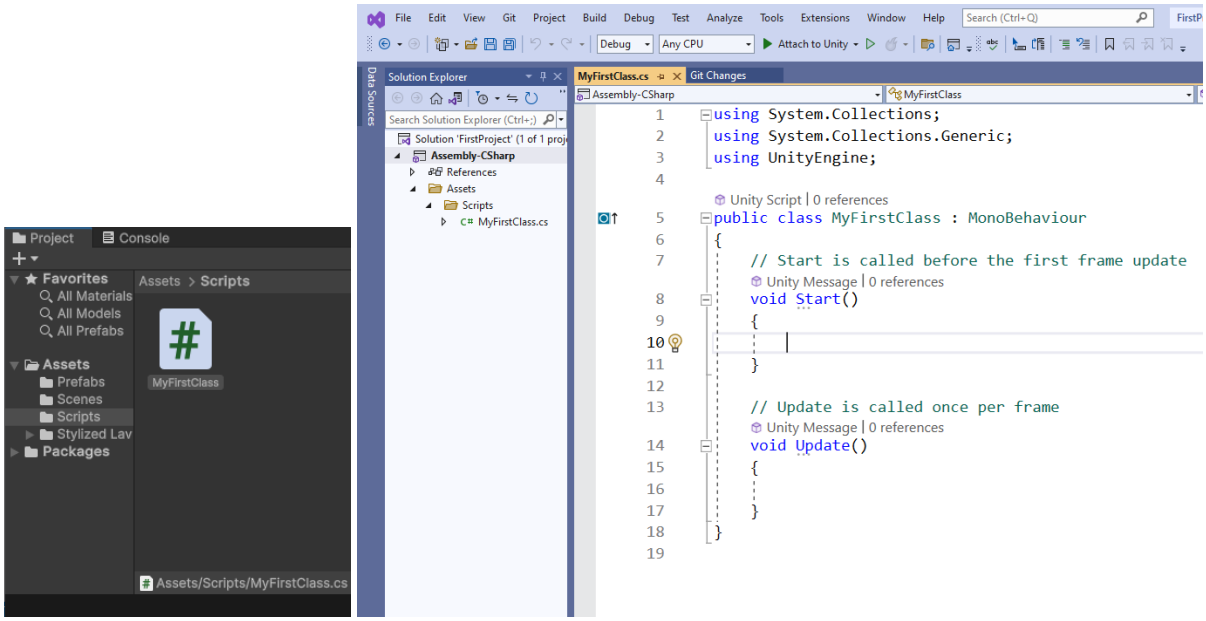
Hình 4.1. Cài đặt Visual Studio giúp Unity hỗ trợ viết lệnh C#

Để mở mặc định mã lệnh bằng công cụ Visual Studio (hoặc Visual Studio Code), người dùng cần thiết lập trên Unity bằng cách vào **Edit**, chọn **Preferences**, trong mục **External Tools**, chọn Microsoft Visual Studio 2022 tại External Script Editor, tích vào 2 mục là **Embedded Packages** và **Local Packages** (Hình 4.2).

Để tạo ra mã lệnh và gán cho một đối tượng trên Unity, trước hết, cần tạo một thư mục trong Project có tên là Scripts (có thể tạo tên thư mục khác). Click phải lên thư mục, chọn Create, chọn C# Script, đặt tên cho tập tin và nhấn Enter. Mở tập tin bằng cách nhấp đôi chuột, Unity sẽ mở tập tin bằng công cụ Visual Studio 2022.



Hình 4.2. Thiết lập để Unity mở mặc định trên bộ Visual Studio 2022



Hình 4.3. Tạo tập tin MyFirstClass.cs và mở bằng Visual Studio 2022

Lưu ý: Tên tập tin phải giống với tên của lớp, nếu không Unity sẽ coi là lỗi mã nguồn và không nhận được lệnh.

2. Viết và gắn lệnh cho đối tượng

Khi tạo mới mã lệnh, Unity tạo sẵn 1 lớp với 2 hàm mặc định là Start() và Update():

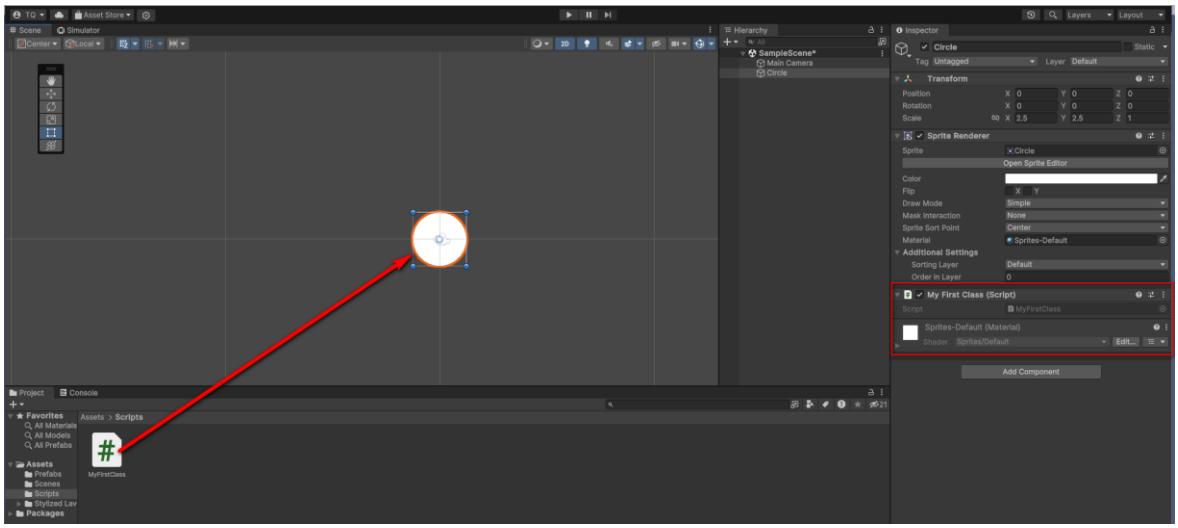
- Hàm **Start()**: Dùng để khởi tạo các giá trị, khi bắt đầu game chạy, sẽ chạy vào hàm này trước, và các giá trị biến, hàm nếu muốn được khởi tạo sẽ được viết trong hàm này.

- Hàm **Update()**: Được dùng để cập nhật giá trị theo từng frame (khung), nghĩa khi người dùng viết lệnh vào hàm này, các giá trị sẽ được cập nhật khi Game chạy qua từng frame.

Ngoài hai hàm mặc định này ra, Unity còn có các hàm khác như FixedUpdate() để cập nhật giá trị theo thời gian, các hàm quản lý va chạm… Các hàm này sẽ được lần lượt giới thiệu ở các phần sau.

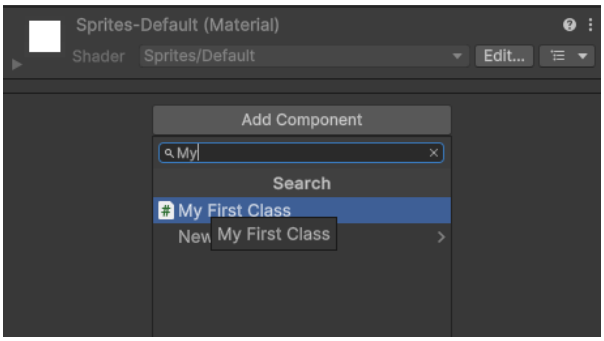
Để gắn mã lệnh cho đối tượng, có 2 cách:

Cách 1: Lựa chọn đối tượng cần gắn trong phần Hierarchy, nhấp chuột vào tập tin mã lệnh, kéo và thả lên Inspector. Hoặc có thể kéo tập tin C# vào thẳng đối tượng trên Scene (Hình 4.4).



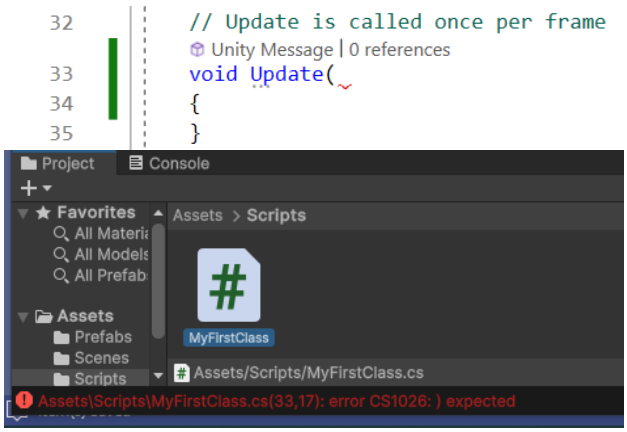
Hình 4.4. Gắn tập tin mã nguồn lên đối tượng

Cách 2: Chọn đối tượng, trong phần Inspector, chọn Add Component, gõ tên tập tin của mã lệnh.



Hình 4.5. Thêm mã lệnh vào trong thành phần của đối tượng Game

Lưu ý là khi mã lệnh viết sai, nếu gắn vào trong Unity, thì sẽ bị báo lỗi (Hình 4.6)



Hình 4.6. Unity báo lỗi sai khi viết lệnh sai tại hàm Update()

3. Kiểu dữ liệu trong C#

Unity cho cho phép khai báo các kiểu dữ liệu cơ bản (của ngôn ngữ lập trình C#) và khai báo thêm các kiểu dữ liệu khác như GameObject, Text, Vector… thể hiện như trong Bảng 4.1

Bảng 4.1. Các kiểu dữ liệu trong C# (Nguồn: unity3d.com)

Kiểu dữ liệu	**Mô tả**	**Ví dụ**
Float	Cho phép khai báo số thực như 0.5 hoặc 13.25.	`private float x = 0.5f;`
Integer	Cho phép khai báo số nguyên, ví dụ 5 hoặc 100.	`private int y = 10;`
Boolean	Lưu trữ giá trị true hoặc false	`private bool f = true;`
String	Lưu trữ chuỗi ký tự như tên hoặc thông báo	`private string str = "Hello everyone";`
Char	Lưu trữ ký tự đơn, kiểu này thường ít sử dụng	`private char c = 'a';`
Enums	Là một tập các giá trị, thường dùng trong kiểu dữ liệu cho phép chọn lựa	`public enum ForceMode { Force = 1, Impulse = 2, }`
Vectors	Dùng để lưu trữ 1 vector với các giá trị X, Y, Z: - *Vector 2*: Chỉ có chiều X và Y trong tọa độ 2 chiều; - *Vector 3*: Có giá trị X, Y và Z trong hệ tọa độ 3 chiều; - *Vector 4*: Là vetor với X, Y, Z và hệ tọa độ W	`Vector2 vt2 = new Vector2(10, 20);`
GameObject	Dùng để lưu trữ một đối tượng Game, ví dụ Sprite, Audio… GameObject thường chứa các thuộc tính như Transform hoặc các thành phần trong đối tượng Game được gắn	`public GameObject gameObj;`
Lists	Là một danh sách các lớp hoặc đối tượng. Danh sách các đối tượng này được dùng như một ArrayList, có các hàm để xử lý dữ liệu như thêm, xóa, đếm số phần tử…	`public List<GameObject> L;`
Dictionaries	Là một danh sách với mỗi phần tử có 2 thành phần là key và value. Ví dụ, có thể tạo ra một danh sách các thành phần với name (key) và age (value).	`Dictionary<int, int> D = new Dictionary<int, int>();`
Object	"Object" là một kiểu đặc biệt, nó có thể gắn cho bất kỳ đối tượng Game nào, ví dụ Audio, Text, Transform,…	`private Object gameObj;`

4. Khai báo và sử dụng biến

Biến trong lập trình C# được sử dụng như là một tham số đầu vào để cho phép người dùng thay đổi giá trị. Biến có hai loại: *Biến cục bộ* là biến khai báo và chỉ sử dụng trong lớp đó, không cho phép người dùng thay đổi; *Biến toàn cục* cho phép người dùng thay đổi bên giao diện Unity.

Ví dụ: Khai báo một biến *speed (tốc độ)* với kiểu truy xuất là private như sau trong lớp MyFirstClass vừa tạo:

```
Unity Script (1 asset reference) | 0 references
public class MyFirstClass : MonoBehaviour
{
    private float speed = 0;
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }
```

Hình 4.7. Khai báo biến cục bộ với từ khóa private

Lúc này biến *speed* là biến *cục bộ*, nó chỉ có hiệu lực bên trong lớp này và không thể can thiệp từ bên ngoài. Để biến *speed* thành toàn cục, có 2 cách như Hình 4.8:

- **Cách 1**: Chuyển private thành public

- **Cách 2**: Thêm chỉ thị *SerializeField* lên trước lệnh khai báo biến

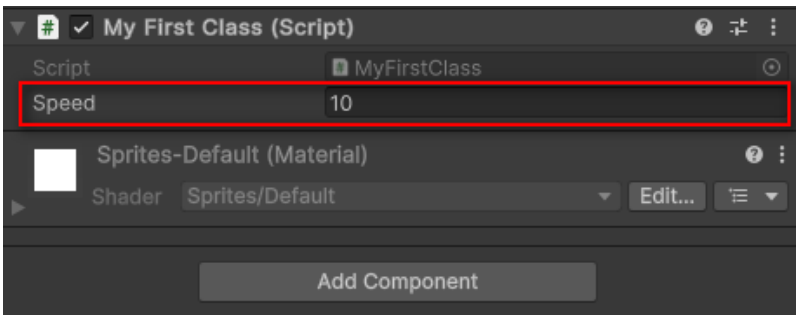
```
Unity Script (1 asset reference) | 0 references
public class MyFirstClass : MonoBehaviour
{

    //public float speed = 0; Cách 1
    [SerializeField]
    private float speed = 0; //  Cách 2
    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {

    }
```

Hình 4.8. Hai cách khai báo biến toàn cục

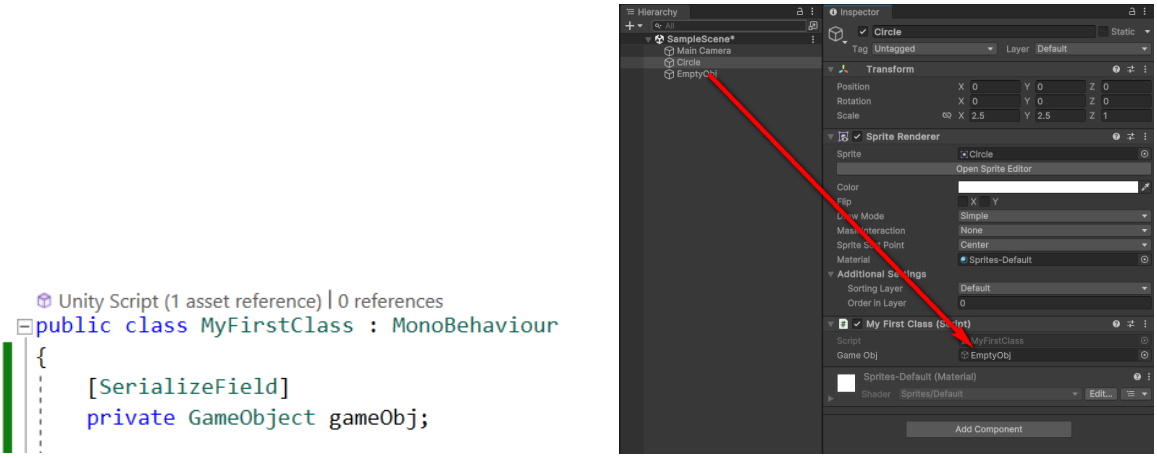
Nếu khai báo biến toàn cục thì biến này sẽ được hiển thị lên bên phần Component của đối tượng Game trong Unity, lúc này người dùng có thể gắn giá trị ban đầu cho biến. Sau khi khai báo như Hình 4.8, lưu lại mã nguồn, sẽ thấy biến hiện lên như trong Hình 4.9. Người dùng có thể thay đổi các giá trị bằng cách gõ vào hoặc dùng chuột kéo và rê chuột như các tham số thường thấy trên Unity Component.



Hình 4.9. Biến toàn cục sẽ hiện lên bên phần Component của đối tượng Game

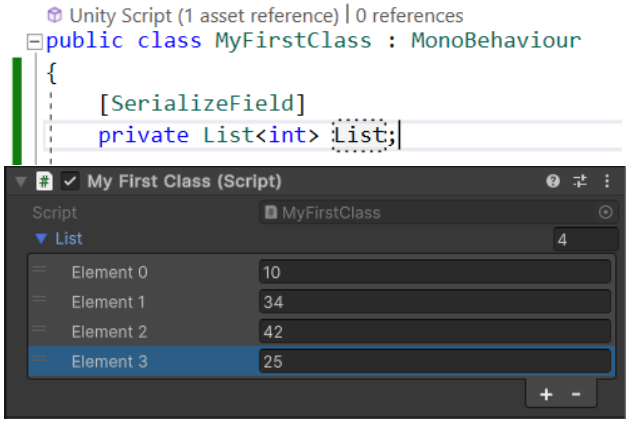
Một số kiểu tham số đặc biệt khác như sau:

- Kiểu ***tham số dạng GameObject***: Khi khai báo kiểu dữ liệu này, trên Unity có thể đưa vào một đối tượng là GameObject bằng cách kéo một đối tượng trên Scene hoặc trên Hierarchy (Hình 4.10).



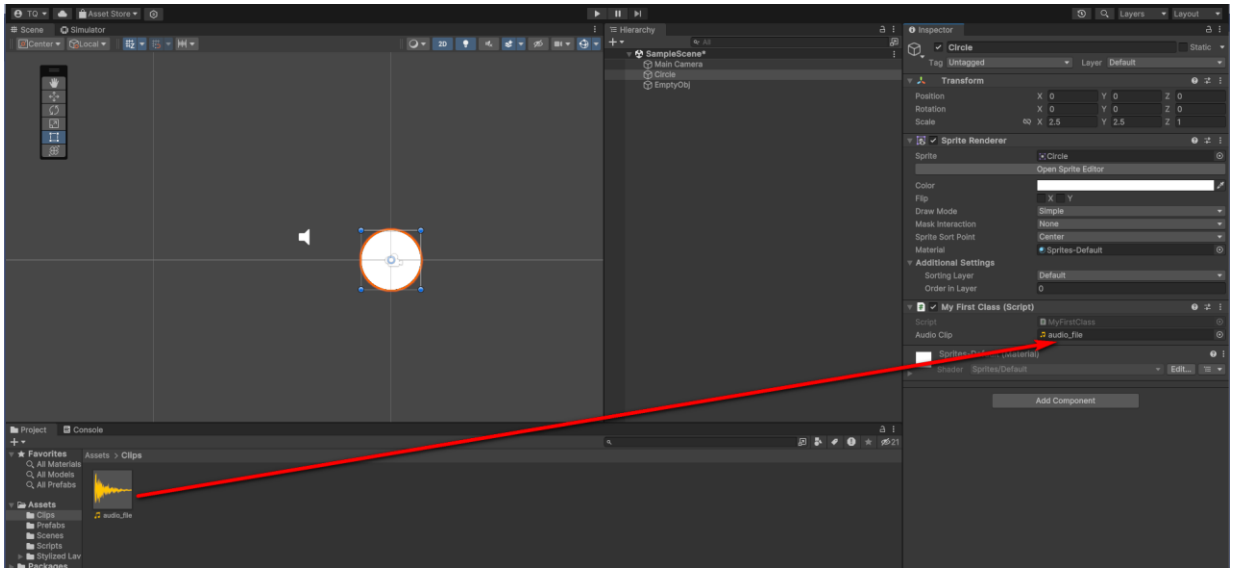
Hình 4.10. Khai báo đối tượng Game Object và thực hiện kéo một EmptyObj để khởi tạo

- Kiểu ***tham số là một danh sách***: Khi khai báo một danh sách, người dùng có thể thêm phần tử cho danh sách bằng cách nhấn vào dấu +, kiểu danh sách có thể là kiểu nguyên thủy hoặc kiểu Game Object (Hình 4.11).



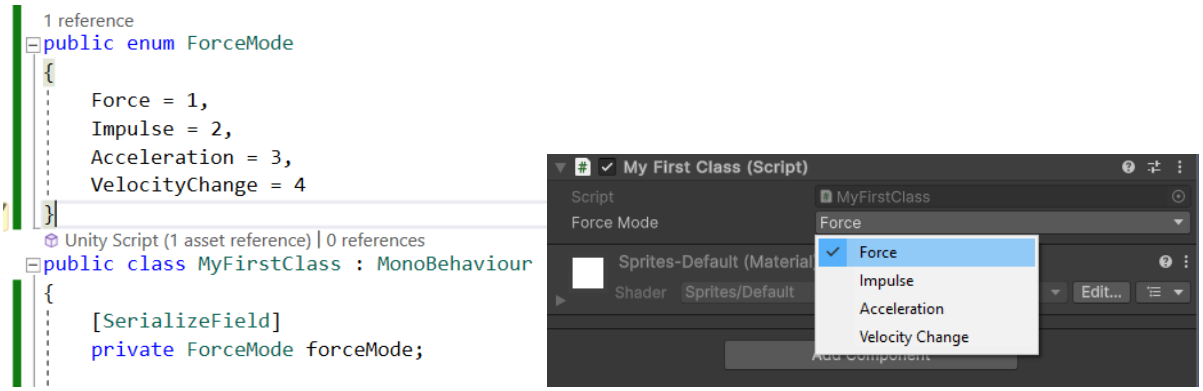
Hình 4.11. Khai báo tham biến kiểu List và thêm vào một tập các phân tử

- Kiểu ***tham số là một tập tin***: Nếu khai báo kiểu tham số là một tập tin, khi muốn truyền tham số, chỉ cần kéo tập tin đó vào tham số. Ví dụ, nếu tham số là một AudioClip thì chỉ cần kéo tập tin audio trong Project vào để gán giá trị ban đầu (Hình 4.12).



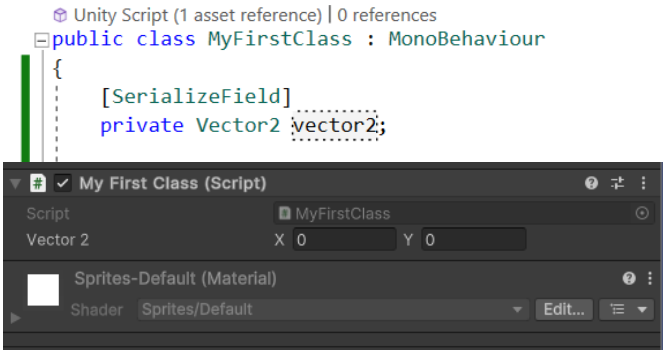
Hình 4.12. Kiểu tham số là một thành phần AudioClip

- Kiểu ***tham số là một enum***: Kiểu này sẽ hiện một ComboBox để người dùng lựa chọn các giá trị (Hình 4.13).



Hình 4.13. Tạo một danh sách lựa chọn bằng kiểu Enum

- Kiểu ***tham số là một Vector***: Nếu khai báo kiểu tham số là Vector2 hoặc Vector3, hệ thống sẽ hiện lên các giá trị X, Y, Z tương ứng với loại Vector đó để người dùng điền giá trị.



Hình 4.14. Kiểu tham số là một Vector2

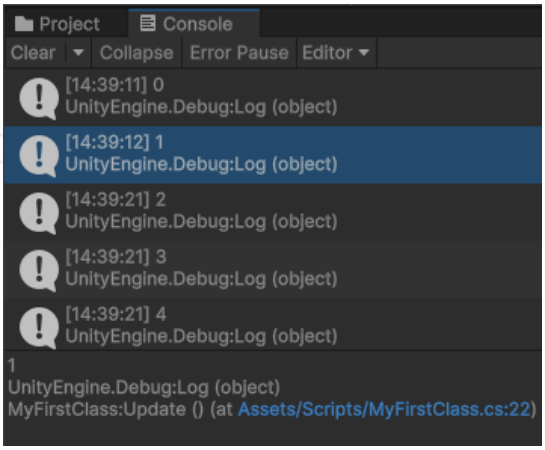
5. Một số phương pháp xử lý mã lệnh

Sau đây là một số phương pháp xử lý mã lệnh cơ bản trong Unity mà người ta thường dùng:

- Giao diện ***Console và Error Messages***: Giao diện ***Console*** là giao diện dùng để xuất một số câu lệnh hoặc thông báo ra màn hình. Khi có lỗi sai, Unity sẽ xuất ra lỗi tại cửa sổ ***Error***

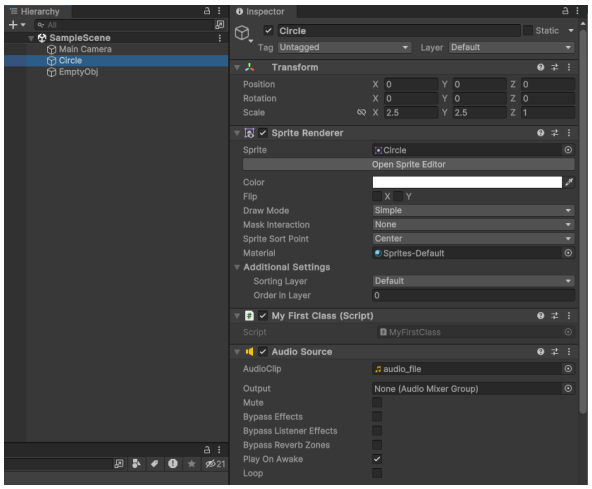
Messages *(xem lại Hình 4.6)*. Lệnh sau đây khai báo một biến ban đầu count = 1, biến này sẽ tăng sau mỗi frame và được xuất ra tại cửa sổ Console (Hình 4.14) sau khi nhấn nút Play.

```
Unity Script (1 asset reference) | 0 references
public class MyFirstClass : MonoBehaviour
{
    private int count = 0;
    Unity Message | 0 references
    void Start()
    {
    }
    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        Debug.Log(count.ToString());
        count++;
    }
}
```



Hình 4.14. Xuất kết quả sau mỗi frame

- Dùng ***hàm GetComponent<> để lấy thành phần***: Để lấy một thành phần nào đó trên đối tượng Game, người ta dùng hàm GetComponent<[tên thành phần]>() và truyền vào tên thành phần cho hàm. Hàm này thường được gọi trong hàm Start(). Ví dụ: Đối tượng Circle có thành phần là AudioSource, trong mã nguồn muốn lấy thành phần này, người ta khai báo một đối tượng AudioSource và lấy bằng hàm GetComponent<>() như trong Hình 4.15.




```
Unity Script (1 asset reference) | 0 references
public class MyFirstClass : MonoBehaviour
{
    private AudioSource audioSource;
    Unity Message | 0 references
    void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }
```

Hình 4.15. Lấy thành phần AudioSource bằng mã lệnh

6. Các hàm và sự kiện trong Unity

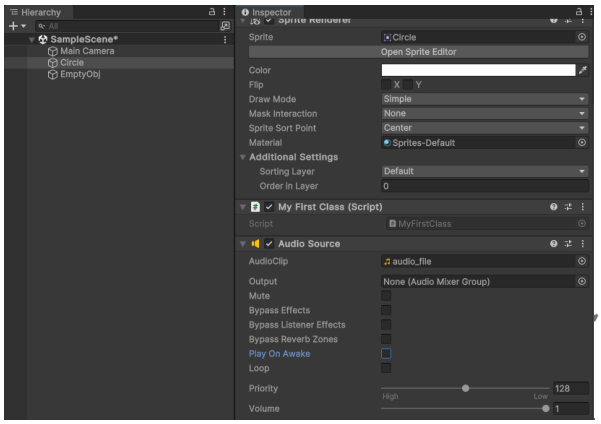
- ***Xử lý phím mũi tên***: Unity hỗ trợ tự động phím mũi tên trái, phải lên xuống bằng khái niệm "*Horizontal*" và "*Vertical*". Để sử dụng phím mũi tên trái, phải, lên xuống, có thể dùng lệnh Input.GetAxis() hoặc Input.GetAxixRaw(). Ví dụ sau đây sẽ làm cho đối tượng di chuyển qua trái, phải, lên xuống khi người dùng nhấn các phím mũi tên trên bàn phím (Hình 4.16).

```
Unity Script (1 asset reference) | 0 references
public class MyFirstClass : MonoBehaviour
{
    private float speed = 5.0f;
    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        float x = Input.GetAxis("Horizontal");
        float y = Input.GetAxis("Vertical");
        transform.Translate(new Vector2(x,y)*speed*Time.deltaTime);
    }
}
```

Hình 4.16. Cách sử dụng sự kiện bàn phím với các phím mũi tên

Lưu ý: Giá trị *speed* là tốc độ của vật thể, tham số `Time.deltaTime` được thêm vào để đảm bảo vật thể chuyển động theo thời gian mà không phụ thuộc vào tốc độ xử lý frame của CPU (hoặc GPU). Thực hiện viết code như trên, gắn script cho đối tượng game và nhấn phím mũi tên qua trái, phải, lên xuống sẽ thấy đối tượng chuyển động.

- Xử lý phím nhấn: Ngoài các phím mũi tên, Unity còn hỗ trợ người dùng sự kiện nhấn phím bằng cách sử dụng hàm Input.GetKey(), Input.GetKeyDown(), và Input.GetKeyUp() với ý nghĩa như nhấn phím, giữ phím và nhả phím. Giá trị truyền vào trong hàm là KeyCode. Ví dụ trong Hình 4.17 thể hiện việc khi người dùng nhấn phím Space, hệ thống sẽ phát ra một âm thanh (Lưu ý là AudioSource đã được thêm vào Component).




```
Unity Script (1 asset reference) | 0 references
public class MyFirstClass : MonoBehaviour
{
    AudioSource audioSource;

    Unity Message | 0 references
    private void Start()
    {
        audioSource = GetComponent<AudioSource>();
    }
    Unity Message | 0 references
    void Update()
    {
        if (Input.GetKeyUp(KeyCode.Space))
        {
            audioSource.Play();
        }
    }
}
```

Hình 4.17. Cách sử dụng sự kiện bàn phím với trường hợp nhấn phím Space

- ***Khởi tạo đối tượng Prefab*** trong Unity: Khi một đối tượng được đưa về dạng Prefab (xem Chương 3, Mục 6), có thể cho hiển thị dễ dàng bằng mã lệnh bằng cách sử dụng hàm Instantiate. Hàm Instantiate có 3 đối số: Vật cần xuất hiện (thường là một Game Object), vị trí xuất hiện, và phép quay ban đầu. Ví dụ sau đây sẽ làm xuất hiện một đối tượng Prefab là một GameObject khi người dùng nhấn phím Space tại vị trí x = 100, y = 100 (Hình 4.18):

```
[SerializeField]
private GameObject gameObject;
Unity Message | 0 references
void Update()
{
    // Nếu người dùng nhấn phím Space thì đối tượng xuất hiện đối tượng
    // tại vị trí x = 100, y = 100
    if (Input.GetKeyUp(KeyCode.Space))
    {
        Instantiate(gameObject,new Vector3(100,100,0), Quaternion.identity);
    }
}
```

Hình 4.18. Ví dụ về việc sử dụng hàm Instantiate(…) trong Unity khi nhấn phím

- **Xử lý sự kiện va chạm:** Xử lý va chạm thường gắn liền với Collider, Unity cho phép xử lý va chạm với hai khái niệm là "*Collision*" và "*Trigger*".

"*Collision*" nghĩa là va chạm về mặt, khi thêm vào đối tượng game một Collider thì sẽ xảy ra va chạm bề mặt, khi đó hàm được dùng để bắt sự kiện này là hàm *OnCollisionEnter2D(Collision2D collision)*. Ví dụ như trong Hình 4.19 sẽ in ra một dòng "Có va chạm" khi vật thể va chạm với một vật thể khác có tên là MatPhang (lưu ý cả hai vật thể đề phải gắn Collider), tham số *collision* sẽ trả về đối tượng bị va chạm. Và khi kết thúc va chạm thì hàm *OnTriggerEnter2D(Collider2D collision)* được gọi.

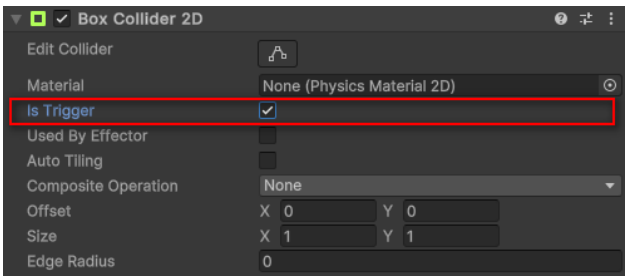
```
Unity Message | 0 references
private void OnCollisionEnter2D(Collision2D collision)
{
    if(collision.gameObject.name == "MatPhang")
        Debug.Log("Có va chạm");
}
Unity Message | 0 references
private void OnCollisionExit2D(Collision2D collision)
{
    Debug.Log("Kết thúc va chạm");
}
```

Hình 4.19. Mã nguồn va chạm bề mặt

Thực hành: Tạo 1 vật thể và một mặt phẳng, gắn RigidBody2D cho đối tượng, gắn thêm Physics Material để vật khi rơi xuống thì nẩy lên. Viết mã nguồn như trong Hình 4.19, chạy và xem kết quả.

"*Trigger*" là va chạm xuyên tâm, xảy ra khi người dùng thêm vào đối tượng game một Collider nhưng click chọn thuộc tính "*Is Trigger*", khi đó vẫn xảy ra va chạm nhưng 2 vật thể sẽ đi xuyên qua nhau, hàm được dùng để bắt sự kiện này là hàm *OnTriggerEnter2D(Collider2D collision)*. Ví dụ như trong Hình 4.20 sẽ in ra một dòng "Bắt đầu va chạm" khi vật thể va chạm và đi xuyên qua một vật thể khác (lưu ý cả hai vật thể đề phải gắn Collider và click chọn *Is Trigger*), tham số *collision* sẽ trả về đối tượng bị va chạm. Và khi kết thúc va chạm thì hàm *OnTriggerExit2D(Collider2D collision)* được gọi.

```
Unity Message | 0 references
private void OnTriggerEnter2D(Collider2D collision)
{
    Debug.Log("Bắt đầu va chạm");
}
Unity Message | 0 references
private void OnTriggerExit2D(Collider2D collision)
{
    Debug.Log("Kết thúc va chạm");
}
```

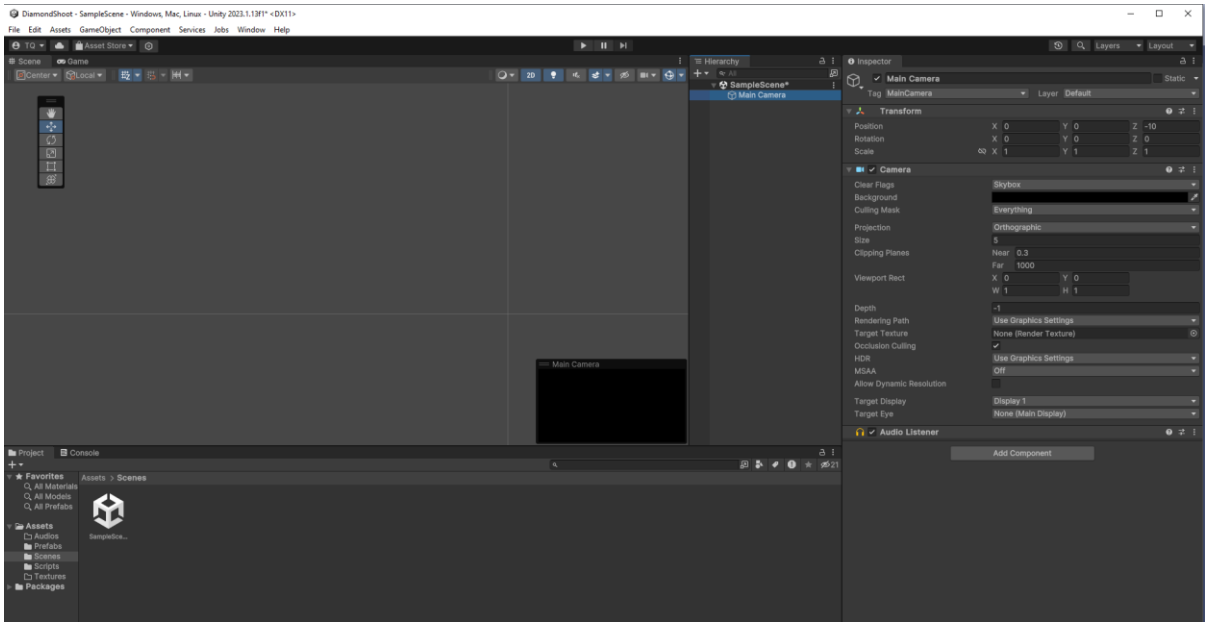


Hình 4.20. Va chạm xuyên tâm với thuộc tính Is Trigger

7. Ví dụ tổng hợp

Hướng dẫn sau đây sẽ tạo ra một kịch bản Game đơn giản bắn kim cương, các loại kim cương sẽ rơi ngẫu nhiên từ trên xuống và người chơi có nhiệm vụ di chuyển khẩu súng để bắn, mỗi lần bắn trúng điểm sẽ tăng lên.

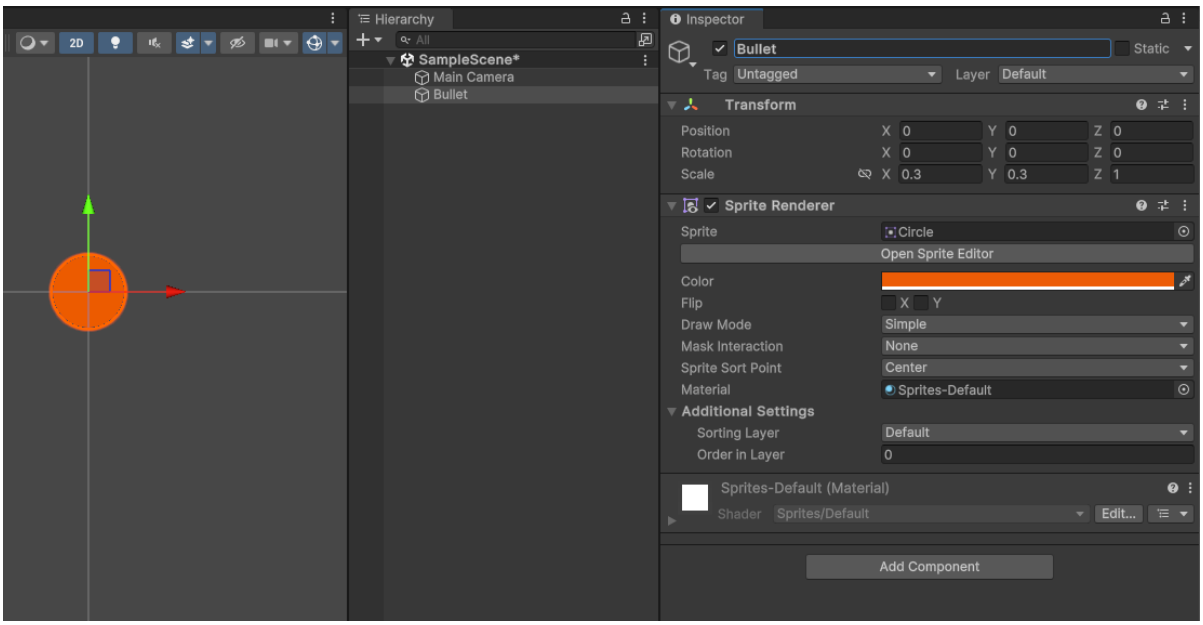
Nhiệm vụ 1: Vào Unity Hub, tạo dự án 2D với tên là DiamondShoot (Hình 4.21). Tạo các thư mục tương ứng trong Project như: Prefabs, Scripts, Textures, Audios… Điều chỉnh màu nền của Camera về màu đen.



Hình 4.21. Tạo dự án DiamondShoot với các thư mục đi kèm

Nhiệm vụ 2: Tạo viên đạn

Tạo ra viên đạn bằng cách vào 2D Object, chọn Sprites, chọn Circle, đặt tên là Bullet điều chỉnh màu và tham số Scale như Hình 4.22.



Hình 4.22. Thiết kế viên đạn

Viết mã nguồn để viên đạn tự bay lên trên khi nhấn nút Play (Hình 4.23). Gắn tập tin Bullet vào viên đạn, nhấn nút Play sẽ thấy viên đạn tự bay khỏi màn hình.

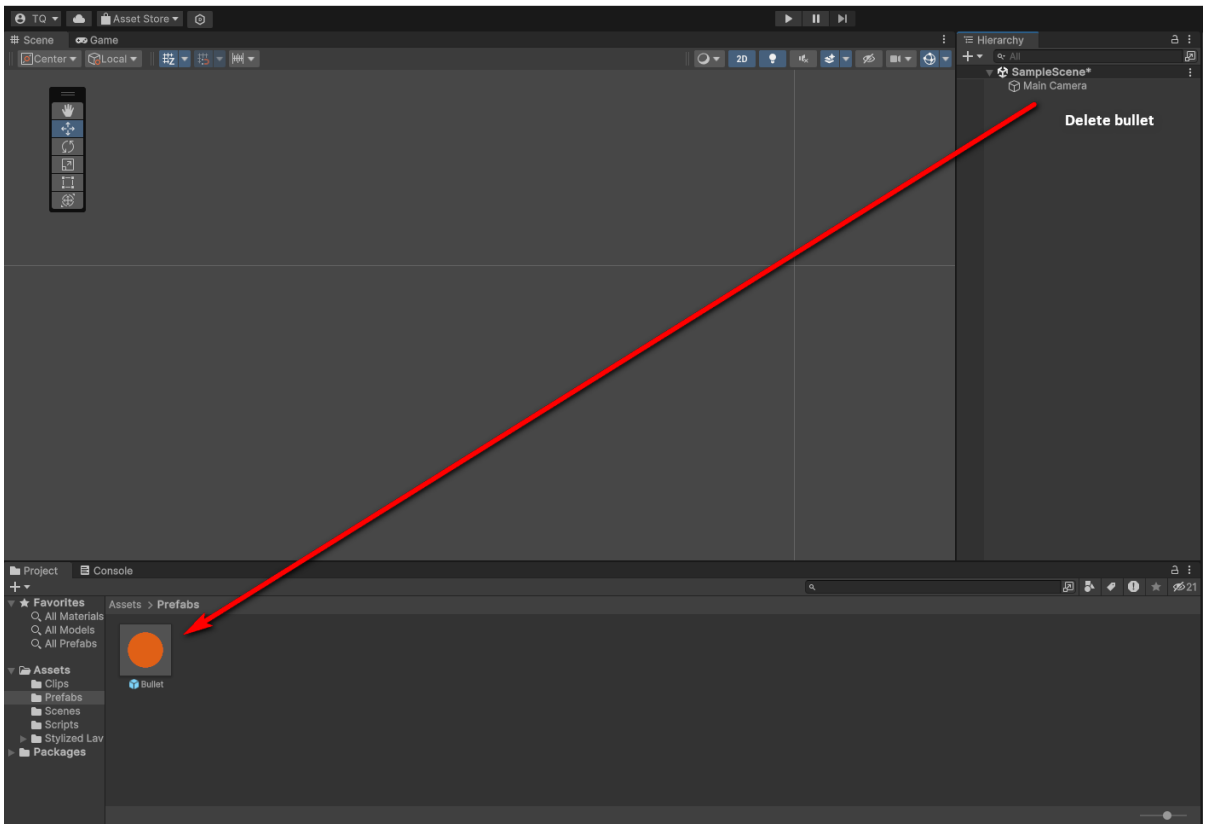
```
Unity Script | 0 references
public class Bullet : MonoBehaviour
{
    // Tốc độ viên đạn là 20
    public float speed = 20.0f;
    // Update is called once per frame
    Unity Message | 0 references
    void Update()
    {
        // Dịch chuyển lên trên với tốc độ đã cho và chỉ phụ thuộc thời gian
        transform.Translate(Vector2.up * speed * Time.deltaTime);

        // Kiểm tra, nếu viên đạn ra khỏi màn hình thì xóa viên đạn
        if (transform.position.y > 5.0f)
            Destroy(gameObject);
    }
}
```

Hình 4.23. Viết script để viên đạn tự bay lên và tự xóa khi ra khỏi màn hình

Lưu ý là trong phần mã nguồn này, hàm Start() đã bị xóa vì không sử dụng đến. Trong hàm Update() viên đạn bay lên trên bằng cách sử dụng hàm dịch chuyển với Vector hướng lên theo tốc độ và không phụ thuộc vào frame. Hàm Destroy() dùng để xóa đối tượng khi nó dịch chuyển ra khỏi màn hình.

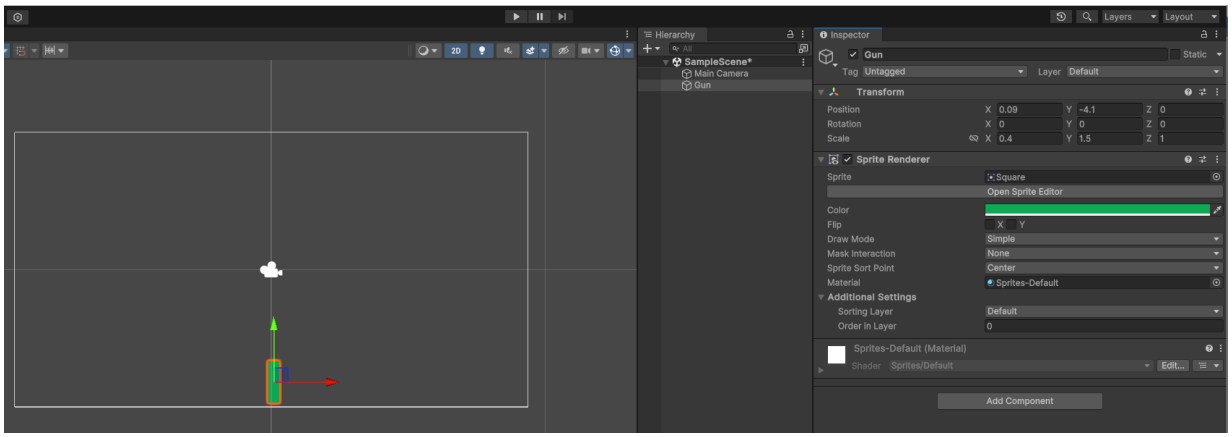
Chuyển Bullet thành Prefab bằng cách kéo xuống thư mục Prefabs trong Project và xóa viên đạn trên Inspector (Hình 4.24).



Hình 4.24. Chuyển Bullet thành Prefab và xóa Bullet trên Hierarchy

Nhiệm vụ 3: Tạo khẩu súng và bắn

Tạo khẩu súng bằng cách vào 2D Object, chọn Sprites, chọn Square, thay đổi kích thước và đặt ở vị trí phù hợp (màu sắc, vị trí, kích thước) (Hình 4.25).



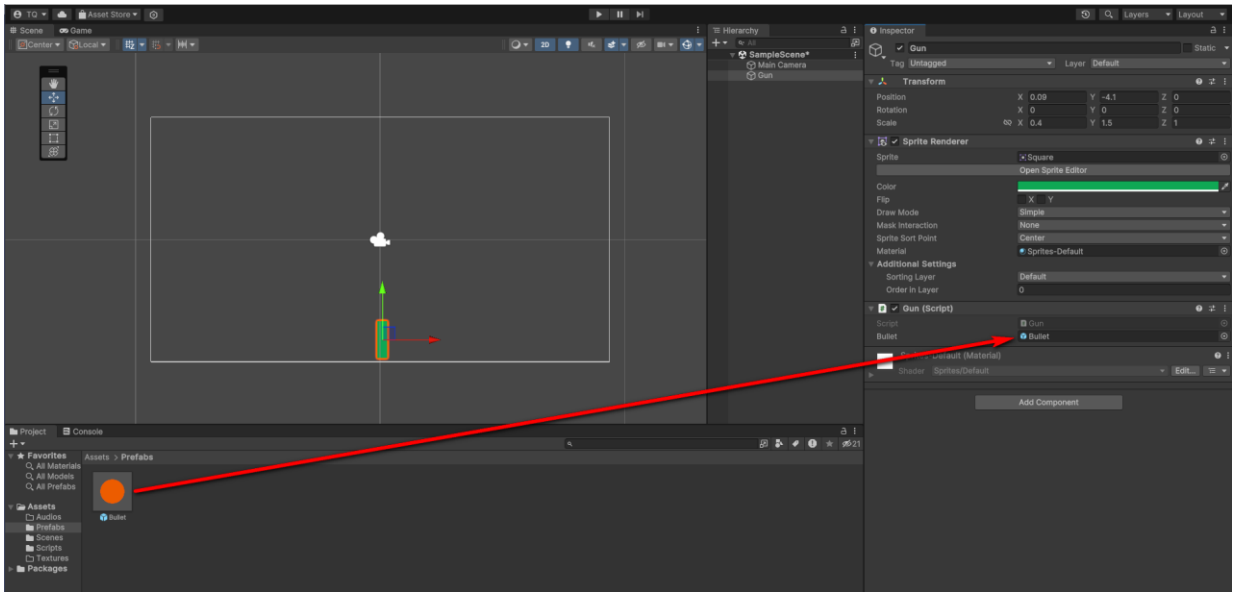
Hình 4.25. Tạo đối tượng Gun, thiết lập các tham số

Viết mã nguồn và gắn vào cho Gun như trong Hình 4.26. Trong mã nguồn này, mỗi lần nhấn phím mũi tên thì khẩu súng sẽ di chuyển qua trái, phải. Khi người dùng nhấn phím Space thì khẩu súng sẽ nhả đạn bằng cách làm xuất hiện đối tượng GameObject.

```
Unity Script | 0 references
public class Gun : MonoBehaviour
{
    // Tốc độ di chuyển của súng
    private float gunSpeed = 5.0f;
    // Viên đạn là một Game object
    [SerializeField]
    private GameObject bullet;
    Unity Message | 0 references
    void Update()
    {
        // Di chuyển súng qua lại trái phải theo tốc độ
        float x = Input.GetAxis("Horizontal");
        transform.Translate(new Vector2 (x, 0)*gunSpeed*Time.deltaTime);
        // Nếu người dùng nhấn phím Space thì cho xuất hiện viên đạn
        if (Input.GetKeyUp(KeyCode.Space))
        {
            // Vị trí xuất hiện viên đạn
            Vector3 v3 = new Vector3(transform.position.x, transform.position.y+0.7f, 0);
            Instantiate(bullet,v3, Quaternion.identity);
        }
    }
}
```

Hình 4.26. Viết mã nguồn cho súng

Gắn mã nguồn cho đối tượng Gun, kéo Bullet từ Prefab và thả vào tham số GameObject của Script này (Hình 4.27)

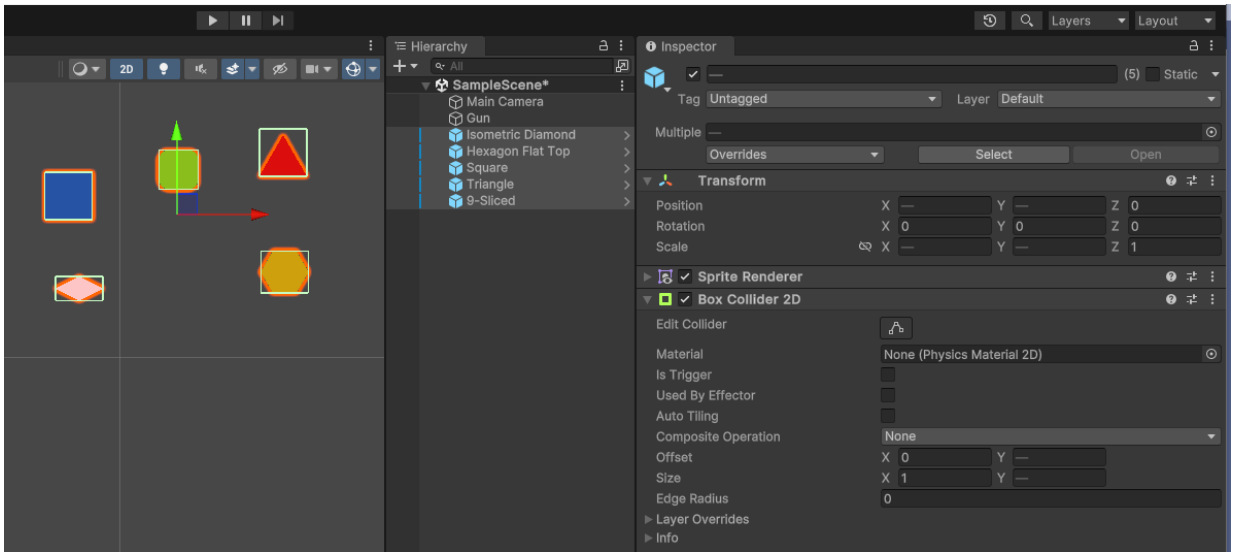


Hình 4.27. Đưa mã nguồn vào đối tượng Gun và gắn Prefab cho GameObject

Nhấn nút Play để chạy và nhấn phím mũi tên để điều khiển khẩu súng di chuyển qua lại, đồng thời nhấn phím Space để bắn đạn, mỗi khi bắn, viên đạn tự di chuyển lên phía trên.

Nhiệm vụ 3: Tạo kim cương

Vào 2D Object, chọn các đối tượng trong Sprites, thay đổi kích thước, màu sắc, vị trí để tạo thành các viên kim cương như trong Hình 4.28. Gắn thêm Box Collider 2D để xử lý va chạm sau này.



Hình 4.28. Tạo các đối tượng kim cương

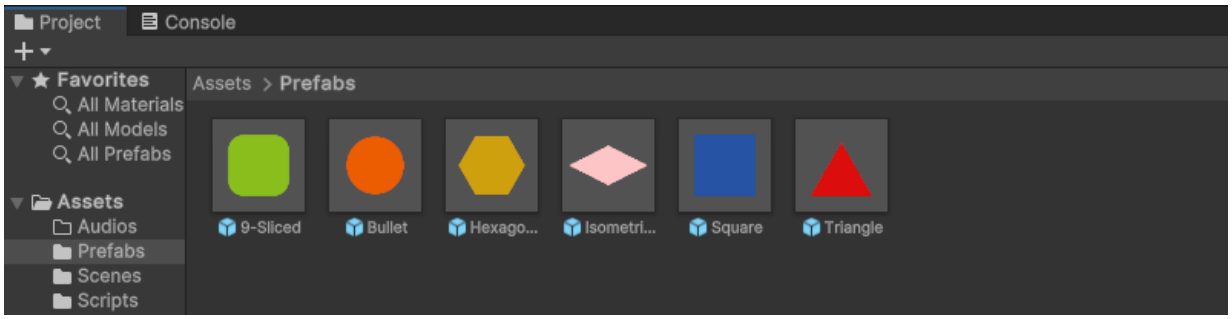
Viết script Diamond và gắn cho từng viên kim cương, lưu ý tốc độ của mỗi viên kim cương sẽ khác nhau tùy thuộc vào hàm Random (Hình 4.29).

```
Unity Script (5 asset references) | 0 references
public class Diamond : MonoBehaviour
{
    // Tốc độ viên kim cương là giá trị ngẫu nhiên
    private int speed = 0;
    Unity Message | 0 references
    private void Start()
    {
        speed = Random.Range(1, 3);
    }

    Unity Message | 0 references
    void Update()
    {
        // Dịch chuyển xuống dưới với tốc độ đã cho và chỉ phụ thuộc thời gian
        transform.Translate(Vector2.down * speed * Time.deltaTime);
        // Kiểm tra, nếu viên kim cương ra khỏi màn hình thì xóa
        if (transform.position.y < -6.0f)
            Destroy(gameObject);
    }
}
```

Hình 4.29. Mã nguồn cho viên kim cương

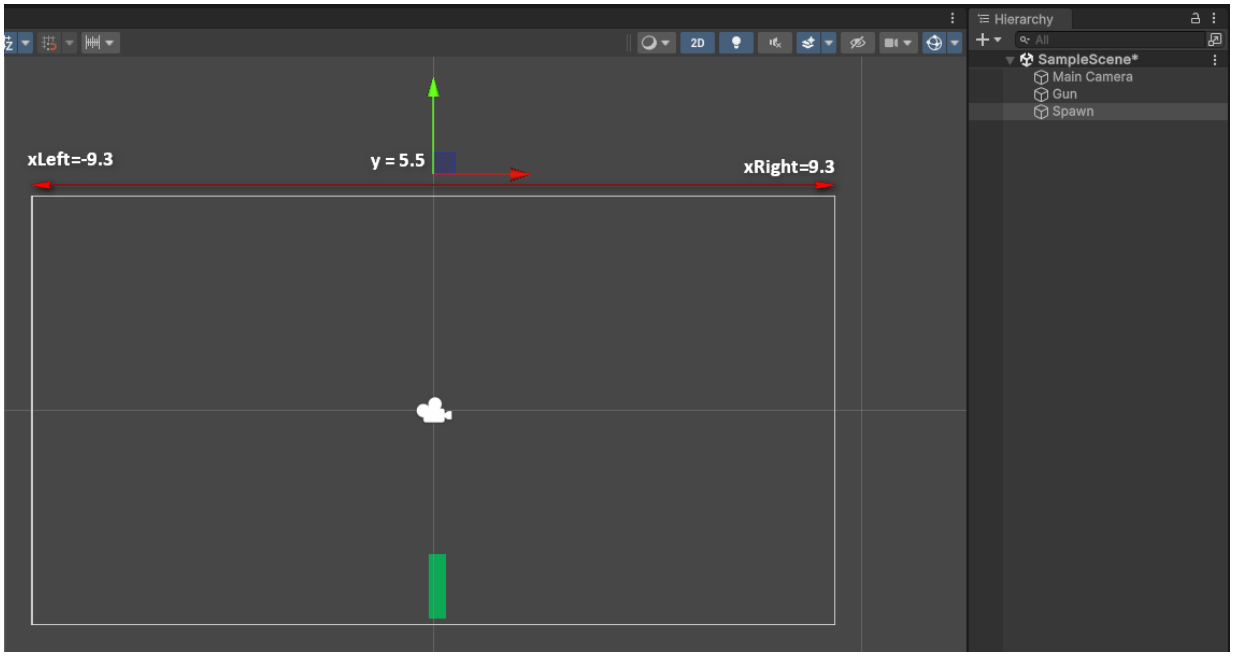
Chuyển 5 viên kim cương này về thư mục Prefabs và xóa chúng trên Hierarchy (Hình 4.30)



Hình 4.30. Chuyển các mẫu kim cương về Prefabs

Nhiệm vụ 4: Làm cho viên kim cương rơi tại các vị ngẫu nhiên

Tạo một đối tượng Empty Object, đặt tên là Spawn phía trên màn hình, ta sẽ lấy vị trí y của đối tượng này để làm vị trí rơi cho viên kim cương. Di chuyển đối tượng Spawn qua trái, hoặc phải để biết tọa độ x của màn hình là từ vị trí nào đến vị trí nào (Hình 4.31). Mục đích của công việc này là cần biết các vị trí để phát sinh các viên kim cương phù hợp.



Hình 4.31. Tạo đối tượng Empty, lấy các giá trị y, xLeft và xRight

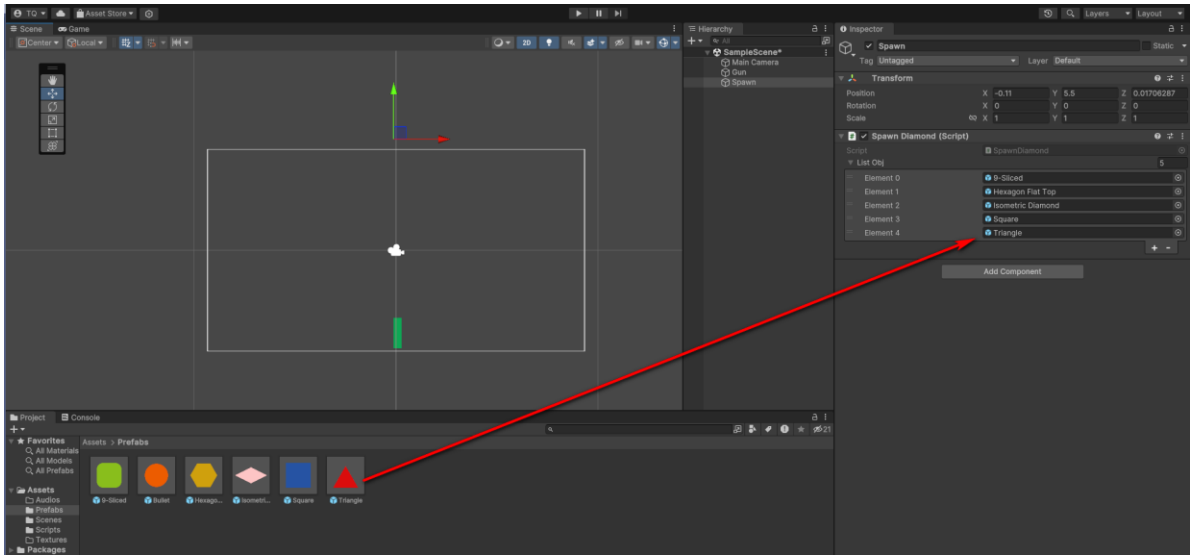
Tạo một script có tên *SpawnDiamond.cs*, viết lệnh như Hình 4.32.

```
Unity Script | 0 references
public class SpawnDiamond : MonoBehaviour
{
    // Khai báo một danh sách các đối tượng Diamonds
    public List<GameObject> listObj;

    Unity Message | 0 references
    private void Start()
    {
        // Hàm này sẽ gọi hàm SpawnRandomDiamond sau một giây và gọi lại nó sau mỗi 2 giây
        InvokeRepeating("SpawnRandomDiamond", 1.0f, 2);
    }
    // Hàm cho xuất hiện ngẫu nhiên viên kim cương
    0 references
    void SpawnRandomDiamond()
    {
        // Phát sinh ngẫu nhiên các chỉ số của mảng
        int ballIndex = Random.Range(0, listObj.Count);
        // Phát sinh vị trí xuất hiện viên kim cương
        float x = Random.Range(-9f, 9f);
        Vector2 pos = new Vector2(x, transform.position.y);
        // Phát sinh viên kim cương ngẫu nhiên tại các vị trí ngẫu nhiên
        Instantiate(listObj[ballIndex], pos, Quaternion.identity);
    }
}
```

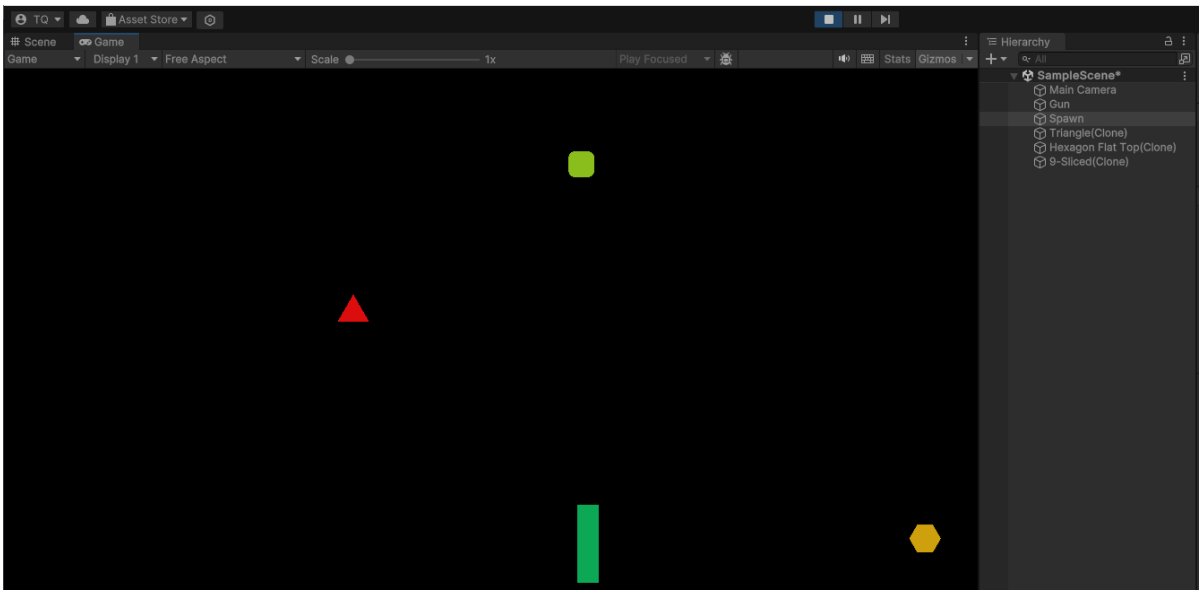
Hình 4.32. Mã nguồn phát sinh ngẫu nhiên kim cương

Quay về Unity, gắn mã nguồn vào đối tượng Spawn, gắn các viên kim cương trong Prefabs vào danh sách tương ứng (Hình 4.33).



Hình 4.33. Gắn các đối tượng kim cương vào danh sách

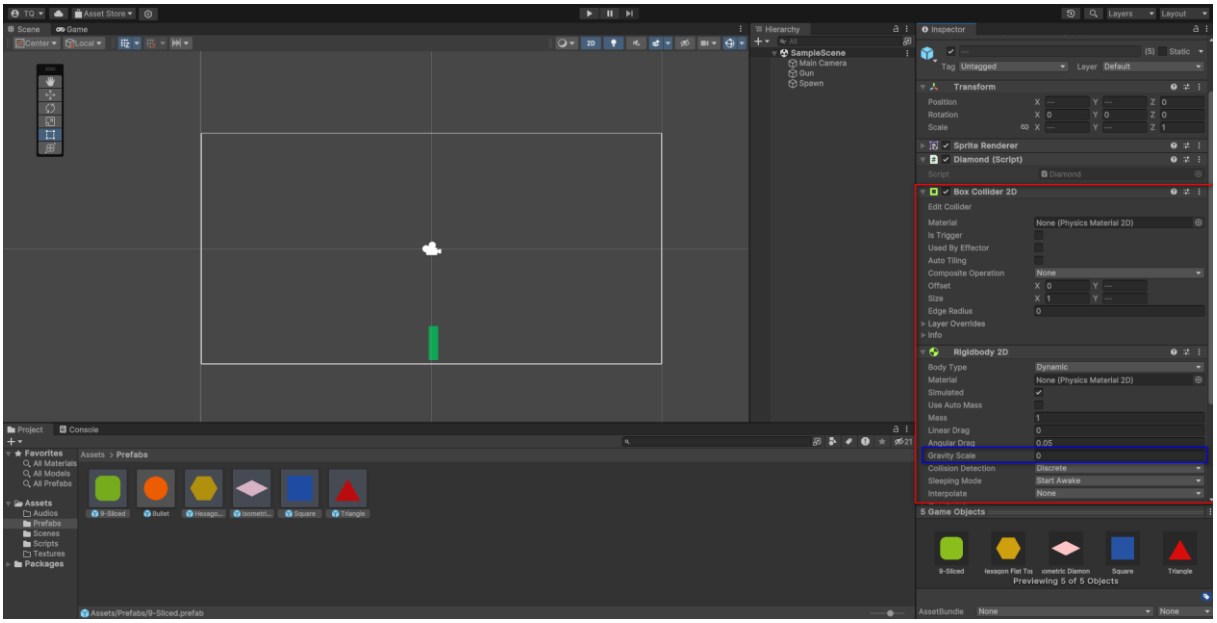
Nhấn nút Play để thấy các viên kim cương rơi ngẫu nhiên như trong Hình 4.34.



Hình 4.34. Kết quả khi nhấn nút Play.

Nhiệm vụ 5: Xử lý va chạm đạn với kim cương

Để xử lý va chạm ta cần thêm Circle Collider 2D cho viên đạn, thêm Box Collider 2D cho tất cả các viên kim cương, đồng thời thêm Rigidbody 2D cho tất cả các đối tượng trên, khi thêm vào cần thiết lập Gravity Scale là 0 (Hình 4.35)



Hình 4.35. Thêm Box Collider 2D và Rigidbody 2D cho viên đạt và kim cương (Gravity Scale=0)

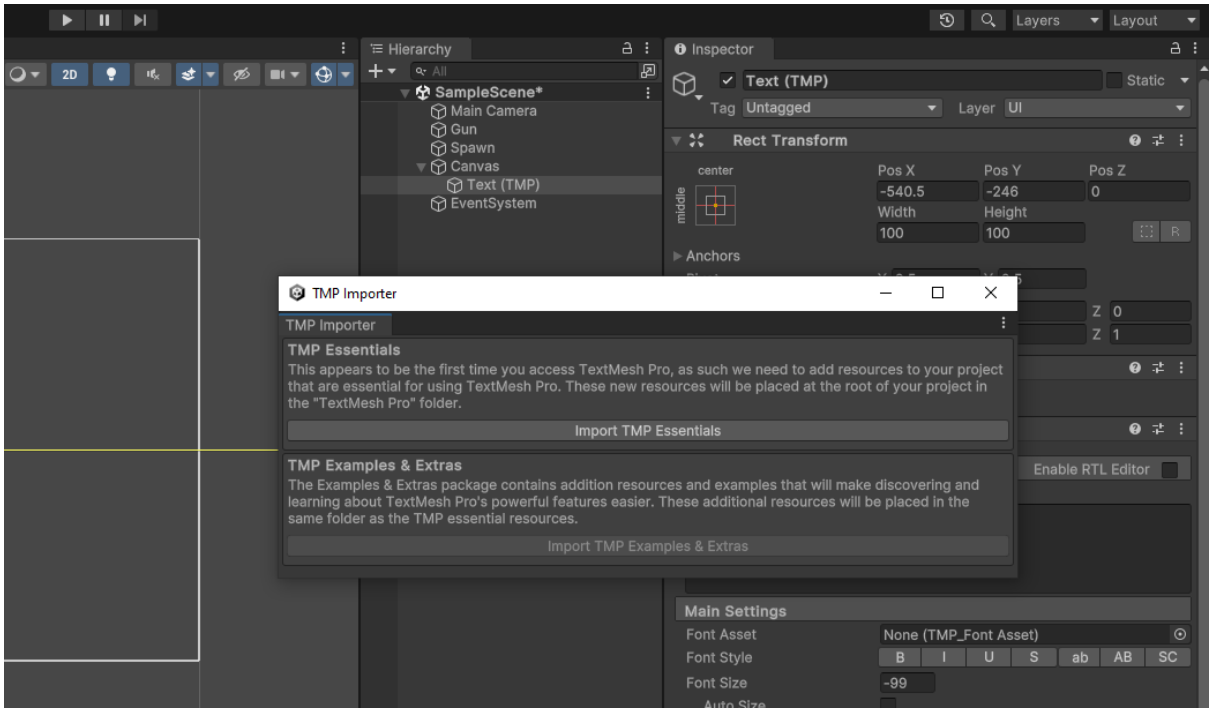
Trong lớp Bullet, thêm hàm *OnCollisionEnter2D(Collision2D collision)* vào như Hình 4.36. Hàm này sẽ được chạy mỗi khi viên đạn va chạm với vất kỳ vật gì. Trong hàm này, sẽ gọi Debug.Log() để báo có va chạm, đồng thời sẽ xóa đi đối tượng bi va chạm (trong ứng dụng này là viên kim cương).

```
    private void OnCollisionEnter2D(Collision2D collision)
    {
        Debug.Log("Có va chạm");
        Destroy(collision.gameObject);
    }
}
```

Hình 4.36. Mã nguồn xử lý va chạm trong lớp Bullet

Nhiệm vụ 6: Hiển thị điểm số mỗi lần bắn trúng (Lưu ý: Phần này là UI, độc giả có thể xem kỹ hơn ở Chương 8)

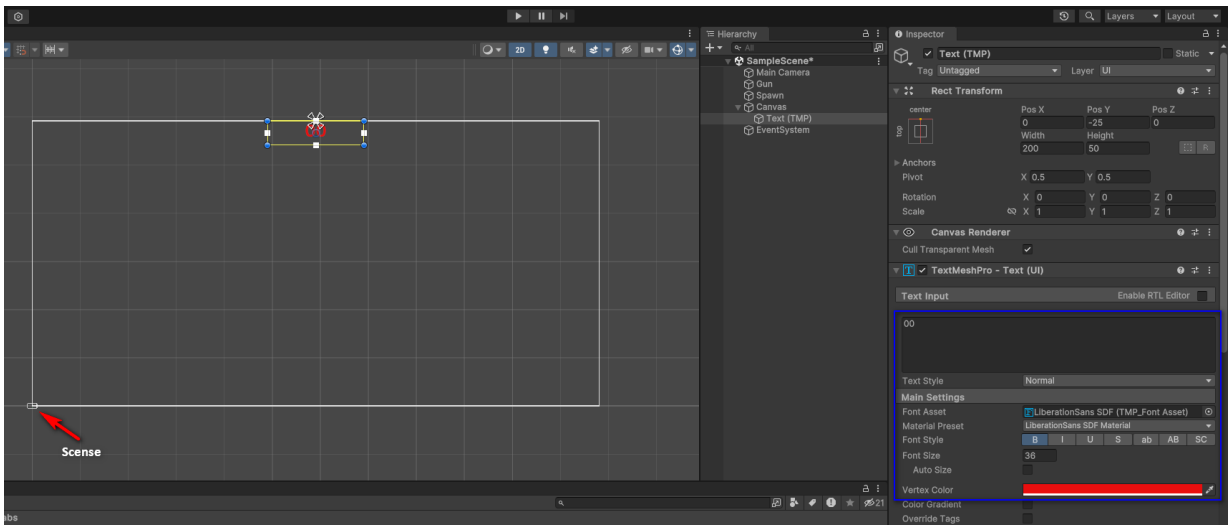
Trên Hierarchy, click phải chọn UI, chọn Text-TextMeshPro, sau đó nhấn nút Import TMP Essentials (Hình 4.37).



Hình 4.37. Thêm vào 1 Text Mesh Pro

Dùng chuột giữa, lăn chuột để thấy rõ được phần UI mà Unity hiển thị. Phần UI thực chất là một Camera khác, được dùng để hiển thị độc lập với Camera chính, khi chạy Game, 2 Camera sẽ chạy cùng lúc, vừa hiển thị game, vừa hiển thị UI.

Sau khi thêm Text Mesh Pro, chỉnh sửa các thông số như vị trí hiển thị, văn bản hiển thị, căn giữa, màu sắc… (Hình 4.38).



Hình 4.38. Thiết lập một số tham số cho Text Mesh Pro

8. Kết chương

Chương này hướng dẫn sử dụng ngôn ngữ lập trình C# trong Unity với bộ Visual Studio. Để lập trình được ngôn ngữ C#, người dùng cần thao tác với các câu lệnh trên các hàm như Start() hoặc Update(). Bên cạnh đó chương này cũng hướng dẫn cho độc giả cách lập trình một game đơn giản với các đối tượng viên kim cương bằng các mã lệnh phù hợp.

Bài tập

1. Hãy nên cách chạy thiết lập để chạy các lệnh trong Unity bằng bộ Visual Studio hoặc Visual Code.

2. Giải thích vai trò của hàm Start() và hàm Update() khi tạo mới một mã nguồn.

3. Hãy nêu và giải thích các kiểu dữ liệu chính trong C#. Bạn hiểu gì về kiểu dữ liệu Vector2 và Vector 3?

4. Sự khác nhau giữa các cách khai báo biến dạng private, public và private với chỉ thị *SerializeField* là gì?

5. Viết lệnh in dòng "Hello Unity" trong Console.

6. Lệnh *Input.GetAxis("Horizontal")* và *Input.GetAxis("Vertical")* dùng để làm gì? Hãy nêu tác dụng của giá trị Time.deltaTime.

7. Hàm *Instantiate* dùng để làm gì? Hãy lập trình để hiển thị một đối tượng trong Prefab lên màn hình khi nhấn phím Enter.

8. Giải thích sự khác nhau giữa hai hàm: *OnCollisionEnter2D(Collision2D collision)* và *OnTriggerEnter2D(Collider2D collision).* Xây dựng ví dụ để minh họa hai hàm này.

9. Thực hiện game *DiamondShoot*. Cải tiến để có nhiều loại kim cương và những loại khác rơi xuống.

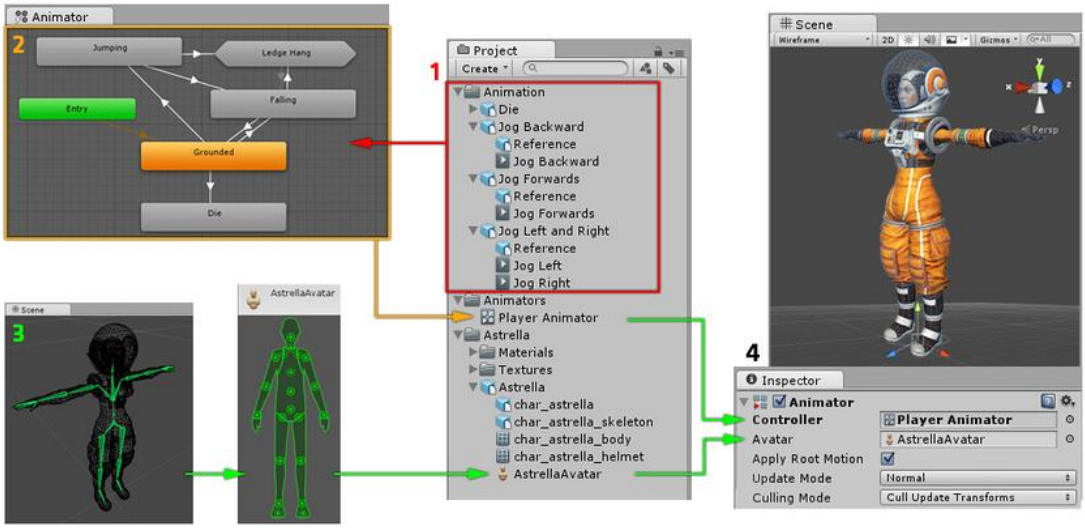
Chương 5. Chuyển động và hiệu ứng

Chương này sẽ hướng dẫn người đọc thực hiện các thao tác chuyển động của các đối tượng game, đồng thời giới thiệu cách tạo hiệu ứng trong game. Đây là hai nội dung quan trọng trong lập trình game, giúp game thêm sinh động và hiệu quả.

1. Giới thiệu

Khi lập trình game, một yếu tố quan trọng làm nên các cảnh sinh động trong game chính là hiệu ứng và chuyển động. Trong Unity hiệu ứng được thiết kế dưới dạng các hệ thống hạt (Particle System), còn hiệu ứng chuyển động được thiết kế dạng Animation.

Bản chất của Animation chính là tạo ra các frame nối tiếp nhau, trong quá trình đó, thay đổi các vị trí hoặc kích thước của đối tượng, lặp đi lặp lại sẽ tạo ra chuyển động (Hình 5.1). Hệ thống hạt chính là sự phối hợp và thay đổi thông số giữa các hạt để tạo nên hiệu ứng.



Hình 5.1. Minh họa mô hình chuyển động trong Unity (Nguồn: docs.unity3d.com)

Hình 5.2. Một số hệ thống hạt cơ bản trong Unity (nguồn developer.valvesoftware.com)

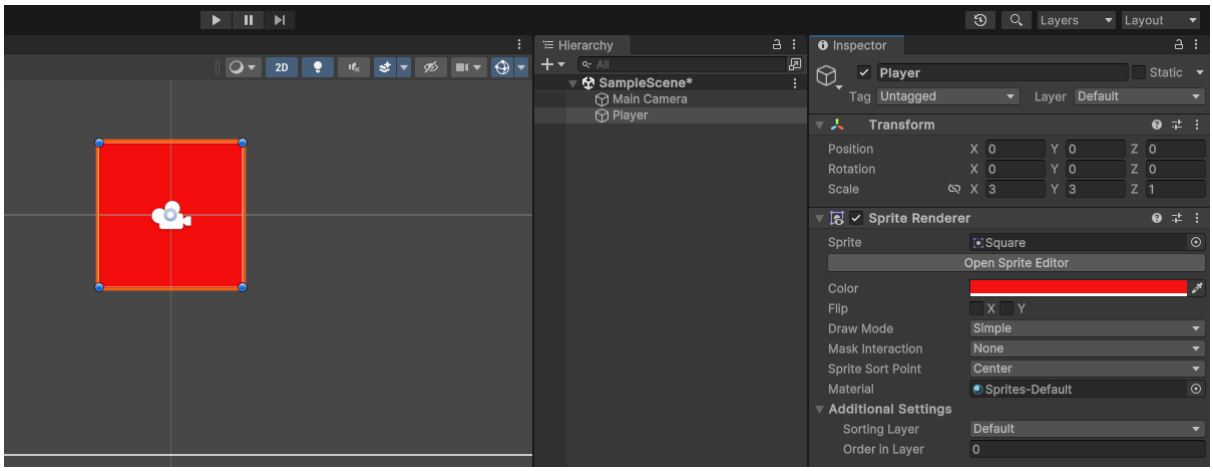
Chương này sẽ giới thiệu một số tham số cơ bản trong cách xây dựng hiệu ứng chuyển động cho đối tượng và tạo các hiệu ứng cơ bản bằng hệ thống hạt.

2. Tạo chuyển động bằng thẻ Animation

Các loại hiệu ứng chuyển động trong Game bao gồm các hiệu ứng như Idle (đứng yên), Run (chạy), Attack (tấn công)… Các loại hiệu ứng này được tạo ra bằng cách tạo ra các tập tin .anim. Tất cả các hiệu ứng sẽ được đặt dưới một bộ quản lý gọi là Animation Controller. Unity cho phép tạo ra các chuyển động bằng các chức năng như Animation (chuyển động), Animator (máy trạng thái), Animation Controller (điều khiển chuyển động).

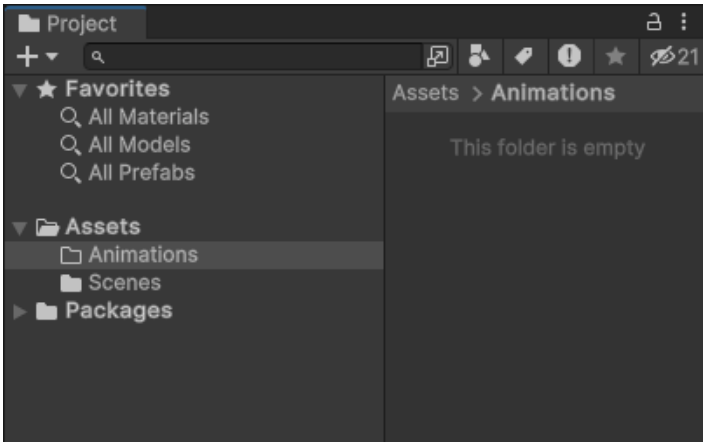
Quá trình sau đây sẽ tạo các hiệu ứng chuyển động lên một đối tượng là Hình chữ nhật màu đỏ. Quá trình này bao gồm các bước như sau:

Bước 1: Tạo đối tượng cần thực hiện hiệu ứng, là một hình chữ nhật màu đỏ đặt tên là Player tại vị trí mặc định, thiết lập thuộc tính Scale là (3,3,1) (Hình 5.3).



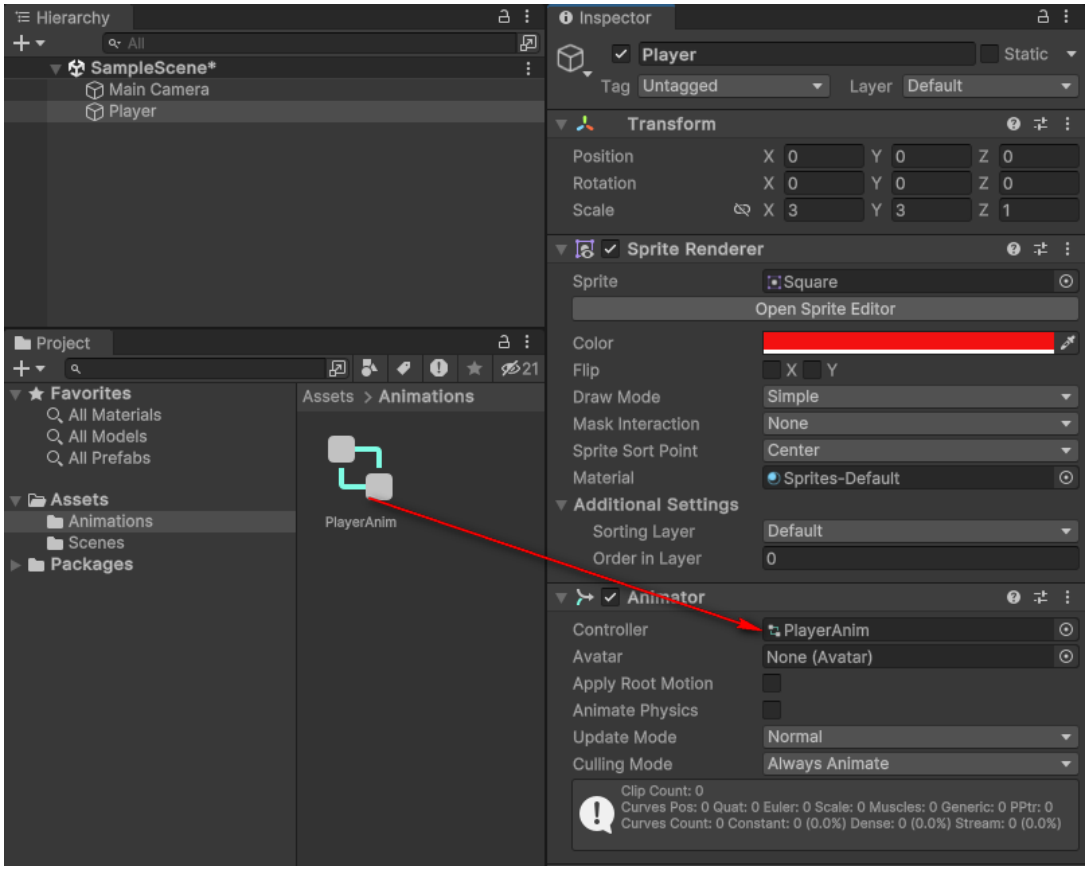
Hình 5.3. Tạo hình chữ nhật trên giao diện

Bước 2: Tạo thư mục Animations trong Project để lưu trữ bộ điều khiển hiệu ứng (Animation Controller) và các hiệu ứng (Animation).



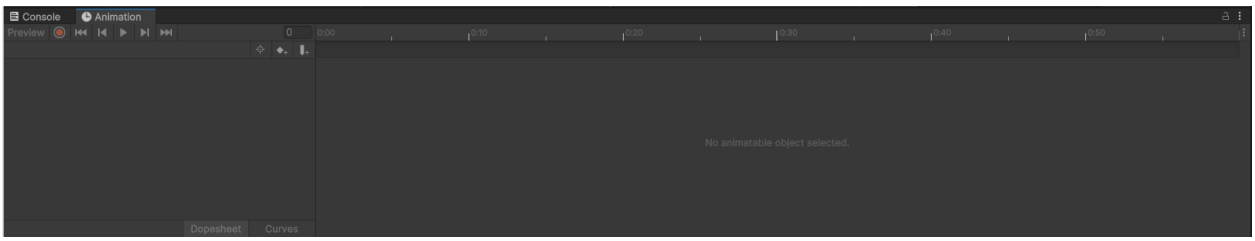
Hình 5.4. Tạo thư mục để chứa các hiệu ứng

Bước 3: Click phải lên thư mục Animations, chọn Create, chọn Animation Controller, đặt tên là PlayerAnim, sau đó kéo vào đối tượng Game vừa tạo. Lúc này đối tượng Game sẽ tự động thêm vào một thành phần là Animator, thành phần này chứa thuộc tính Controller là PlayerAnim vừa tạo (Hình 5.5).



Hình 5.5. Tạo Animation Controller

Bước 4: Trên thanh Menu, vào Window, chọn Animation, Animation (hoặc nhấn Ctrl + 6), thẻ Animation xuất hiện, kéo và đặt nó để nằm cùng thẻ Console (hoặc Project) như Hình 5.6.

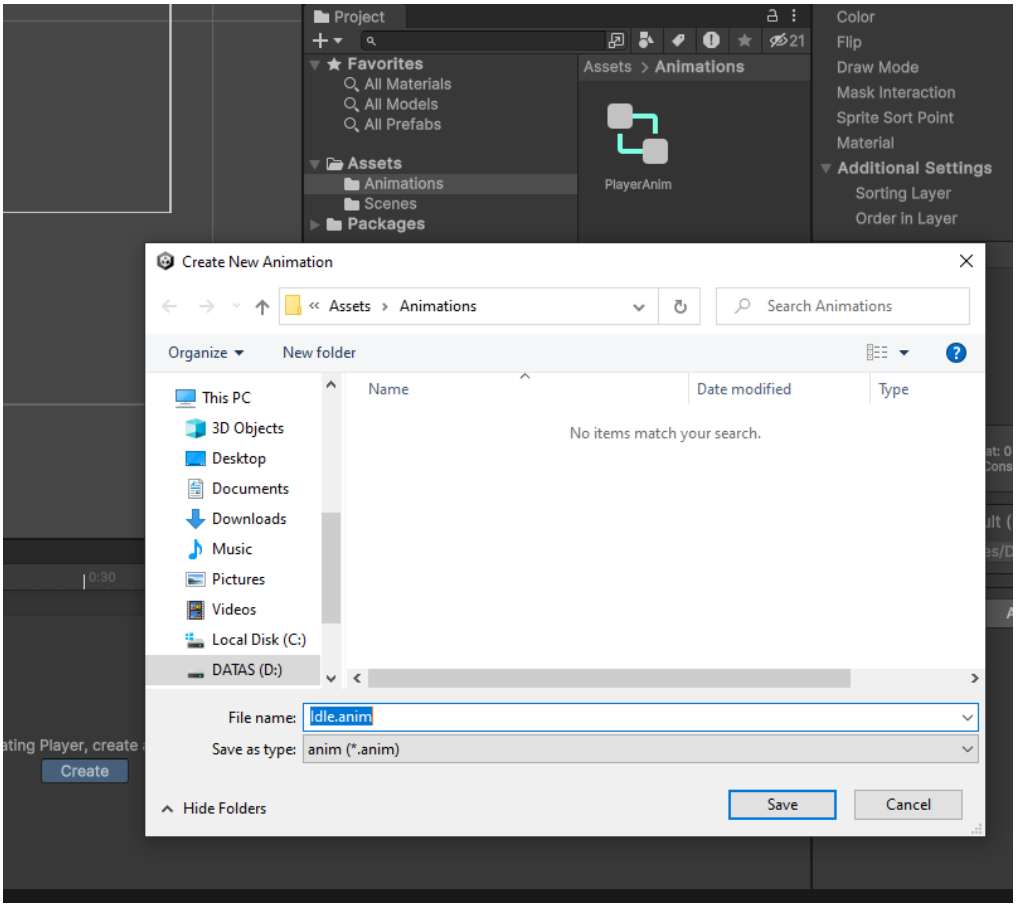


Hình 5.6. Quản lý chuyển động bằng Animation

Trong thẻ Animation, phần bên trái là các đối tượng được tạo chuyển động và kiểu chuyển động, bên phải là thanh TimeLine dùng để thiết lập thời gian trong chuyển động. Các tính năng này đang bị vô hiệu hóa do chưa chọn đối tượng Game và không có chuyển động nào được tạo ra.

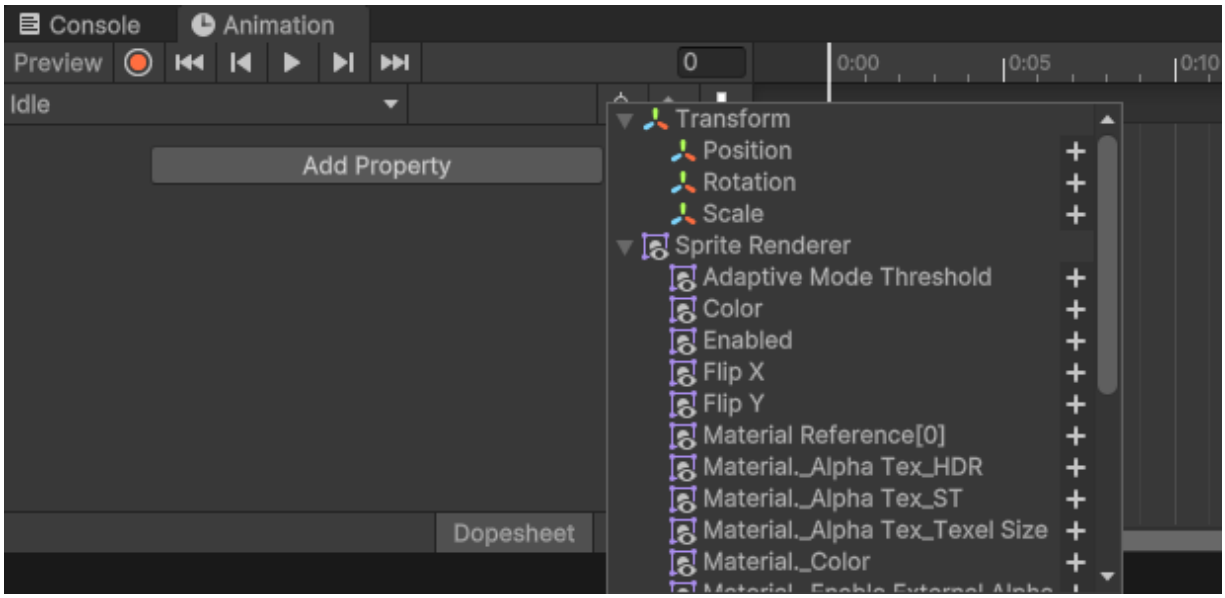
Bước 5: Tạo chuyển động

Để tạo chuyển động, trước tiên cần chọn đối tượng Game (là hình vuông đỏ ở Bước 1) sau đó nhấn nút Create (trong thẻ Animation), đặt tên là ***Idle.anim*** và lưu vào một thư mục Animations (Hình 5.7).



Hình 5.7. Tạo mới một tập tin chuyển động

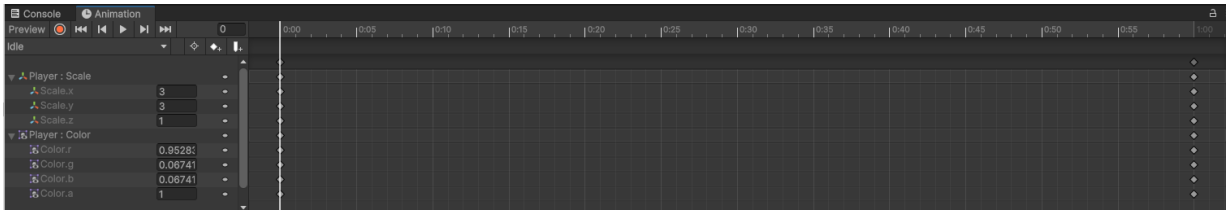
Bước 6: Để thêm loại hiệu ứng cho chuyển động Idle, click vào nút Add Property, có hai loại chuyển động là Transform và Sprite Renderer (Hình 5.8). Transform cho phép thực hiện các chuyển động dựa trên vị trí, quay và co dãn, trong khi đó Sprite Renderer cho phép thực hiện các chuyển động trên giao diện như lật ngược, đổi màu thay đổi chất liệu, vô hiệu hóa…



Hình 5.8. Các loại chuyển động khi nhấn nút Add Property

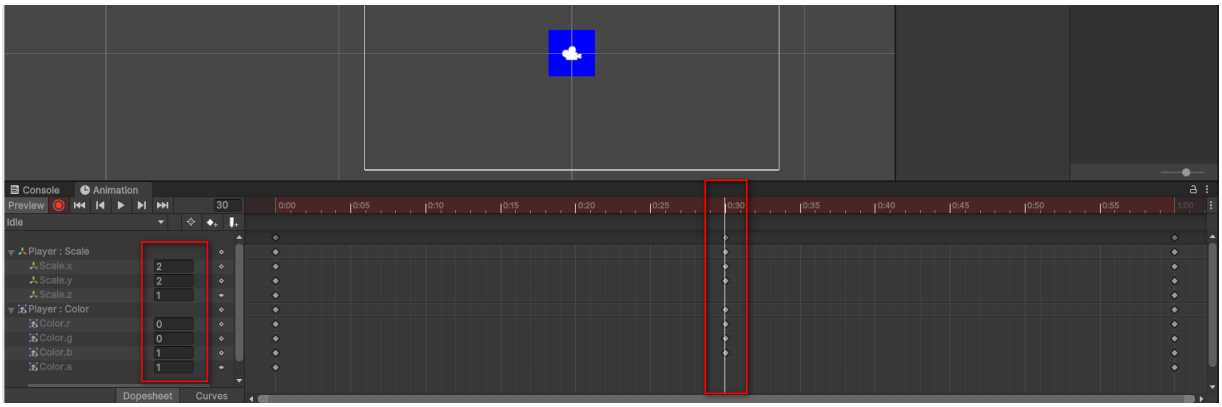
Với trạng thái Idle, ta sẽ cho hình vuông tự phóng to, thu nhỏ và đổi màu. Bấm Add Property, chọn Transform, bấm dấu + trong phần Scale, lúc này khu vực bên trái hiện ra các

tham số dùng để thực hiện co dãn là Scale.x, Scale.y, và Scale.z. Bấm chọn thêm Color trong phần Sprite Renderer (Hình 5.9).



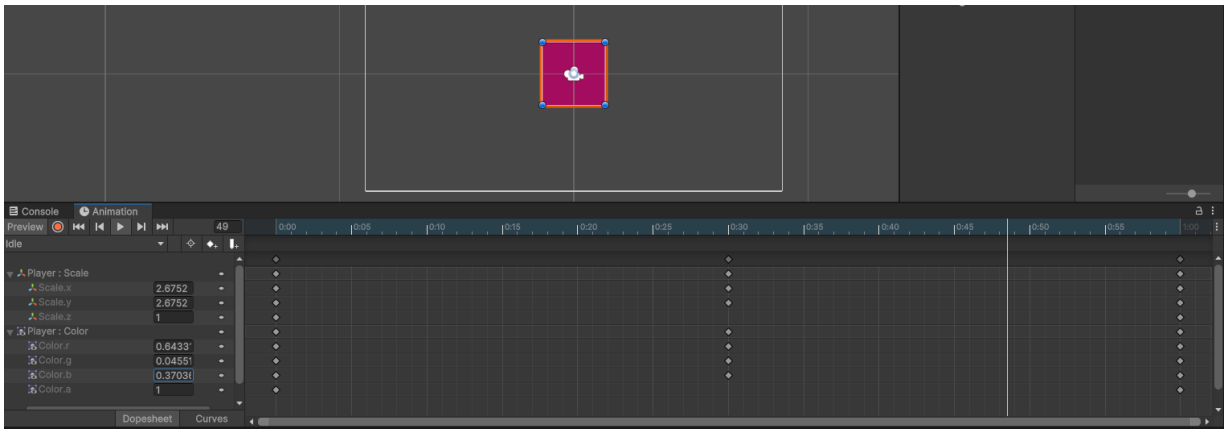
Hình 5.9. Chọn tham số Scale và Color cho đối tượng Game

Trước hết bấm chuột vào nút ⏺ để ghi lại hiệu ứng, bấm chuột chọn ô hình thoi tại vị trí 0:30, nhập vào Scale.x, Scale.y giá trị 2. Tiếp tục đổi màu qua màu xanh tại vị trí này (Hình 5.10).



Hình 5.10. Tạo hiệu ứng tại vị trí 0:30

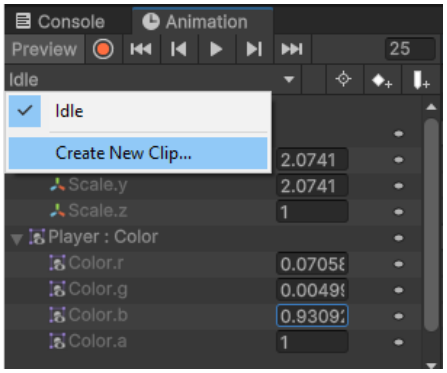
Nhấn nút Play của Animation hoặc nút Play của giao diện Unity thành quả, đối tượng tự động phóng to, thu nhỏ và đổi màu sau mỗi một giây (Hình 5.11). Nếu muốn chậm hơn, có thể kéo dài thời gian bằng cách kéo lùi thanh hình thoi ra xa hơn.



Hình 5.11. Hiệu ứng của trạng thái Idle trong 1 giây

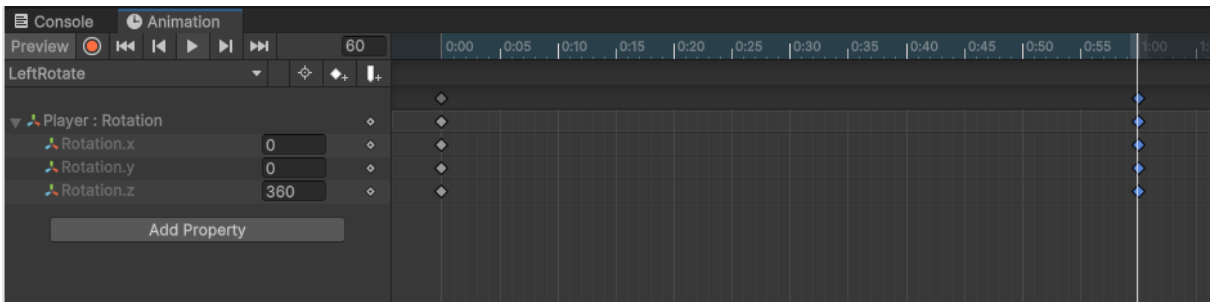
Bước 7: Tạo các clip khác

Trong khu vực bên trái, Animation cho phép tạo ra các Clip Animation khác bằng cách chọn Create New Clip… (Hình 5.12). Trong ví dụ này, ta cần tạo thêm hai clip là Left Rotate (quay trái) và Right Rotate (quay phải).



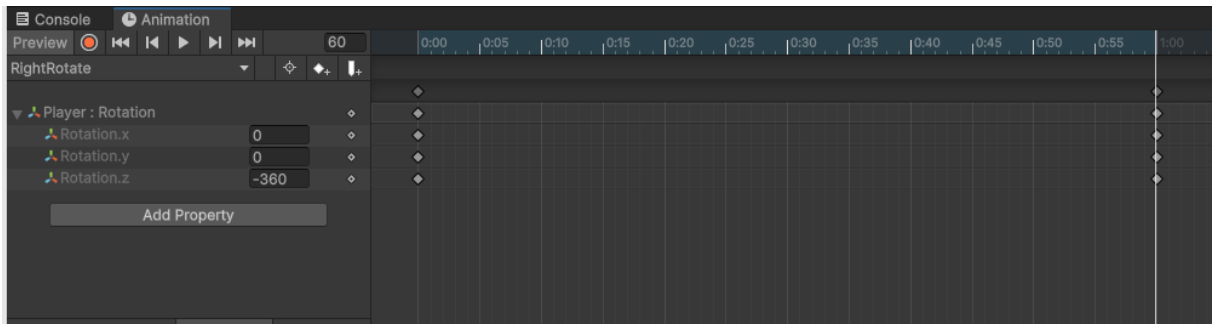
Hình 5.12. Tạo mới một Clip Animation

Clip Left Rotate được tạo ra bằng cách tạo mới một Animation có tên là LeftRotate.anim, thêm vào thuộc tính Rotation, rồi thiết lập để sau 1 giây sẽ quay một vòng qua trái bằng cách tại vị trí 1:00, thiết lập Rotation.z là 360 (Hình 5.13)



Hình 5.13. Tạo Animation quay trái

Thực hiện tương tự cho Animation quay phải RightRotate.anim với việc đổi dấu góc quay(Hình 5.14)

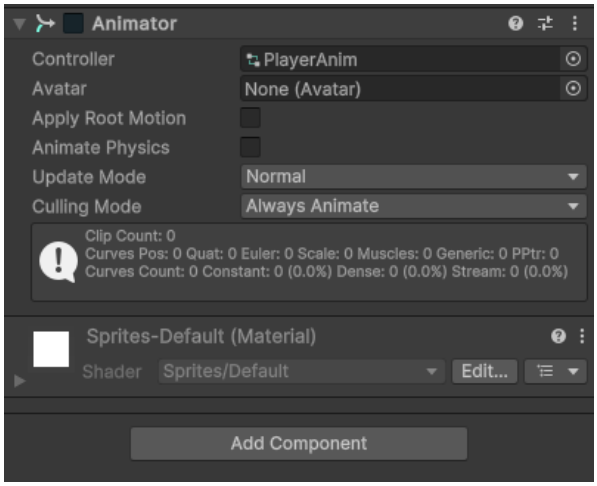


Hình 5.13. Tạo Animation quay phải

Cần lưu ý rằng, một Game có thể có nhiều Clip, dùng để biểu diễn các trạng thái khác nhau của nhân vật như: Đứng yên (Idle), chạy (Run), tấn công (Attack), ngã xuống (Fall)…. Mỗi trạng thái này được tạo ra bằng cách tạo mới một Clip. Việc liên kết qua lại giữa các Clip sẽ được trình bày trong các phần sau.

3. Quản lý chuyển động

Khi tạo mới một chuyển động dạng Clip, hệ thống tự gắn cho đối tượng một thành phần gọi là Animator (Hình 5.14), đây chính là thành phần dùng để quản lý chuyển động. Các thuộc tính chính trong Animator được trình bày như trong Bảng 5.1. Tất nhiên là người dùng có thể thêm vào thành phần này bằng cách chọn Add Component, rồi chọn Animator nhưng phần Controller sẽ được để trống và được thêm vào sau.

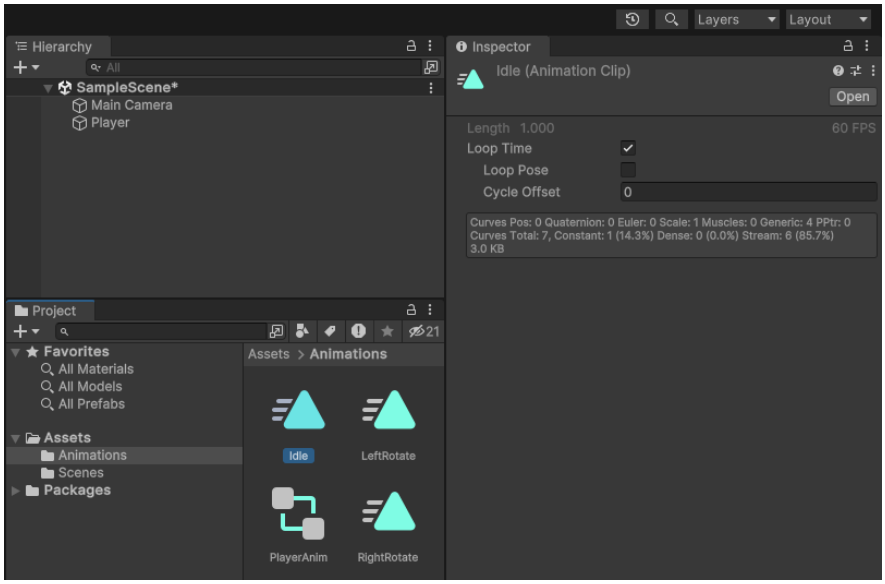


Hình 5.14. Thành phần Animator

Bảng 5.1. Các thuộc tính của Animator

Thuộc tính	**Chức năng**
Controller	Bộ điều khiển chứa các Clip được gắn cho đối tượng
Avatar	Là Avatar của nhân vật, chức năng này chỉ áp dụng cho nhân vật dạng hình người 3D
Apply Root Motion	Chức năng này nếu được chọn sẽ giúp điều khiển nhân vật xoay xung quanh vị trí của nó bằng mã lệnh
Animate Physics	Chức năng này giúp kích hoạt chế độ vật lý của chuyển động
Update Mode	Chức năng này cho phép chọn lựa cách thức cập nhật chuyển động, có 2 cách thức bao gồm cập nhật theo vật lý (Animate Physics) và cập nhật theo thời gian (Unscaled Time)
Culling Mode	Chức năng này cho phép vô hiệu hóa việc chọn các ảnh động

Quan sát trong Project/Assets/Animations sẽ thấy có 4 tập tin được tạo bao gồm 1 tập tin Controller và 3 tập tin dạng Clip là Idle, LeftRotate, RightRotate (Hình 5.15). Tập tin Controller giúp điều khiển hiệu ứng hoạt động, các tập tin Clip chính là trạng thái của nhân vật.



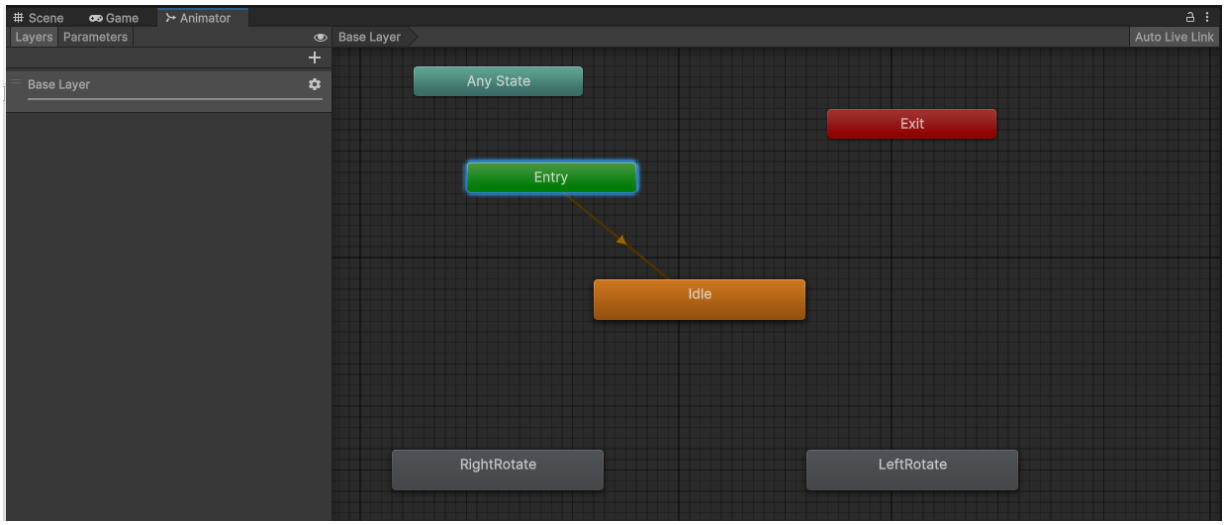
Hình 5.15. Tập tin Controller và 2 tập tin Clips

Khi bấm vào một Clip bất kỳ, trên thanh Inspector sẽ có tham số LoopTime cho phép lặp đi lặp lại các hiệu ứng (Hình 5.15).

4. Quản lý các trạng thái

Tập tin Animation Controller bản chất là một máy trạng thái. Máy trạng thái là một khái niệm dùng để quản lý và điều khiển các trạng thái của đối tượng Game. Khi xây dựng một game đơn giản thì nhân vật (hoặc đối thủ) thường có các trạng thái như đứng yên, chạy, nhảy, tấn công, ngã xuống, chết… Unity quản lý các trạng thái này bằng một khái niệm gọi là Animation Controller hay là máy trạng thái.

Nhấp đúp chuột vào PlayerAnim, Unity sẽ mở các trạng thái lên một thẻ gọi là Animator (Hình 5.16). Đây chính là phần Unity dùng để quản lý các trạng thái của nhân vật.

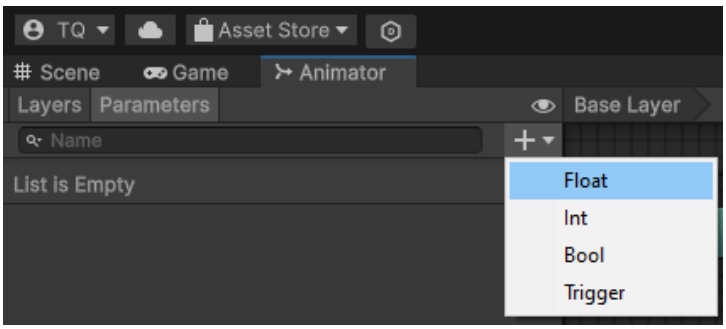


Hình 5.16. Các trạng thái trong thẻ Animator

Màn hình quản lý trạng thái Animator bao gồm các tính năng sau:

- ***Layers***: Dùng để quản lý lớp, chức năng này cho phép quản lý các Layer trong Animator.

- ***Parameters***: Các tham số, phần này dùng để khởi tạo các tham số để quản lý trạng thái, có 4 kiểu tham số là Float (dùng để thiết lập các tham số để lưu trữ số thực), Int (thiết lập các tham số để lưu trữ giá trị nguyên), Bool (dùng để thiết lập các tham số có trạng thái bật-tắt), và Trigger (dùng để kích hoạt một sự kiện) (Hình 5.17). Sau này khi xử lý Animation, cần tham số nào thì sẽ khởi tạo tham số đó, ví dụ khi nhấp chuột thì chuyển về trạng thái bắn (lúc này cần khởi tạo sự kiện IsTrigger).



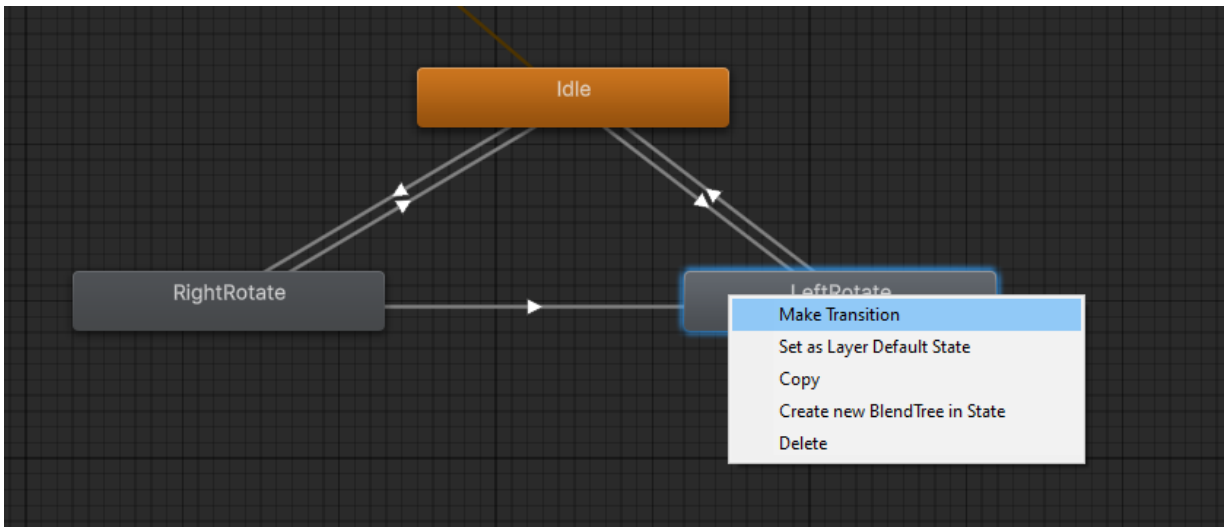
Hình 5.17. Các kiểu tham số trong Animator

- **Giao diện chính (Base Layer)**: Phần này dùng để quản lý trạng thái, bao gồm ba trạng thái đã tạo trước đó (Idle, Rotate, Jump), ngoài ra còn có thêm các trạng thái mặc định như Entry

(là trạng thái ban đầu), Any State (là ám chỉ bất kỳ trạng thái nào), Exit (là trạng thái khi thoát game).

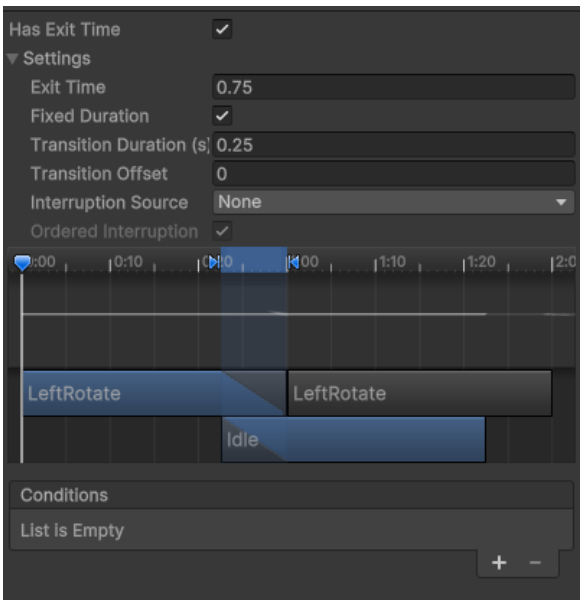
Dễ dàng nhận thấy có một đường nối giữa trạng thái Entry và trạng thái Idle, điều này có nghĩa là khi bắt đầu vào Game, nhân vật sẽ chuyển về trạng thái đứng yên (Idle). Bấm vào một trạng thái bất kỳ, ta có các thông tin chính như sau trong phần Inspector: ***Motion***: Là đối tượng chuyển động mặc định, ta có thể thay đổi đối tượng chuyển động nếu thấy không phù hợp; ***Speed***: Tốc độ chuyển động (mặc định là 1) và một số tham số khác.

Các trạng thái có mối quan hệ qua lại với nhau bằng cách thiết lập các đường nối mũi tên, như ta thấy mặc định ban đầu hệ thống sẽ chuyển từ trạng thái Entry về trạng thái Idle. Để thiết lập dịch chuyển giữa các trạng thái, click phải lên một trạng thái, chọn Make Transition, kéo chuột tới trạng thái cần tới và thả chuột (Hình 5.18). Mũi tên chỉ theo hướng nào thì trạng thái có thể được chuyển dịch theo hướng đó.



Hình 5.18. Thiết lập đường đi giữa các trạng thái

Ngoại trừ mũi tên chuyển trạng thái từ Entry qua Idle, các mũi tên còn lại khi bấm chuột vào đều có các tham số đi kèm như trong Hình 5.19. Các tham số được mô tả như trong Bảng 5.2.



Hình 5.19. Các tham số trong một đường nối trạng thái

Bảng 5.2. Các thuộc tính của đường nối trạng thái

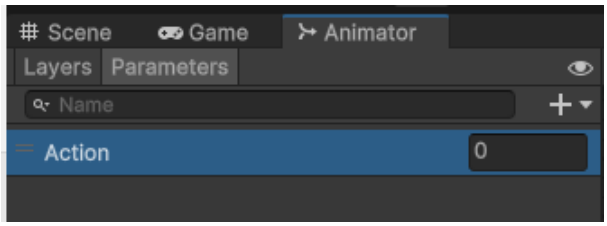
Thuộc tính	**Chức năng**
Has exit time	Tham số này cho phép thiết lập thời gian để kết thúc chuyển một trạng thái qua một trạng thái khác
Settings	Một số thiết lập cho chuyển trạng thái như: Thời gian kết thúc chuyển trạng thái, khoảng thời gian chuyển trạng thái, vùng đệm…
Conditions	Điều kiện chuyển trạng, điều kiện ở đây sẽ phụ thuộc vào danh sách tham số có trong Parameters (xem Hình 5.19). Phần này được giới thiệu ở mục sau
Preview	Giúp nhìn rõ được từng hiệu ứng một cách tách biệt

Lưu ý: Nếu tất cả các trạng thái đều nối với nhau theo mũi tên 2 chiều, lúc này ta có thể làm nhanh hơn bằng cách thiết lập đường nối giữa trạng thái Any State với các trạng thái còn lại.

6. Chuyển trạng thái bằng mã lệnh

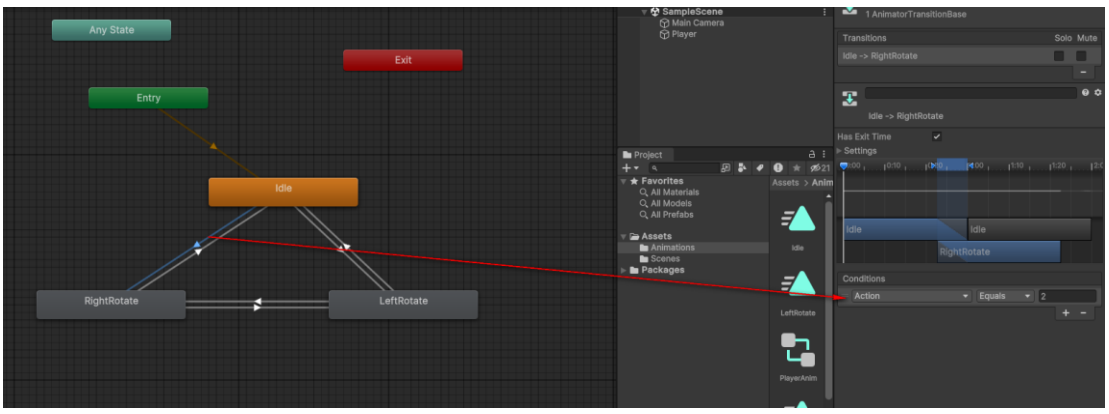
Để chuyển nhân vật từ trạng thái này qua trạng thái khác, ta cần thiết lập bằng mã lệnh, nghĩa là khi người dùng nhấn phím hay bấm chuột thì trạng thái sẽ được chuyển. Trong ví dụ này, giả sử người dùng nhấn phím mũi tên qua trái thì sẽ xoay trái (nhả phím thì về trạng thái đứng yên), tương tự với phím mũi tên phải.

Lúc này cần thiết lập một biến để điều khiển trong phần Parameters của Animator (xem lại Hình 5.17). Giả sử biến kiểu in, có tên là Action, ta quy ước như sau: nếu Action = 0 thì đứng yên, Action = 1 thì quay trái, Action = 2 thì quay phải. Đầu tiên, vào Animator, bấm vào dấu + để tạo biến kiểu Int, đặt tên là Action (Hình 5.20)



Hình 5.20. Tạo biến Action trong Animator

Tiếp theo, trong lược đồ trạng thái của Animator, bấm vào từng mũi tên dịch chuyển trạng thái, qua phần Inspector, click vào dấu + trong phần Conditions, chọn biến Action và thiết lập giá trị tương ứng. Các trạng thái chuyển về RightRotate thì Action = 2 (Equals) (Hình 5.21); Các trạng thái chuyển về Idle thì Action = 0; Các trạng thái chuyển về LeftRotate thì Action = 1.



Hình 5.21. Thiết lập trạng thái khi dịch chuyển từ Idle qua RightRotate

Tạo mới một tập tin script đặt tên là PlayerController.cs, kéo và gắn vào đối tượng Player. Viết mã nguồn như trong Hình 5.22.

```
public class PlayerController : MonoBehaviour
{
    private Animator animator; // Khai báo đối tượng Animator
    private int action = 0; // Biến action
    Unity Message | 0 references
    void Start()
    {
        animator = GetComponent<Animator>(); // Lấy Animator của Player
    }
    Unity Message | 0 references
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.LeftArrow)) { // Nhấn phím mũi tên trái
            action = 1;
            animator.SetInteger("Action", action);
        }
        if (Input.GetKeyUp(KeyCode.LeftArrow)) // Nhả phím mũi trái
        {
            action = 0;
            animator.SetInteger("Action", action);
        }
        if (Input.GetKeyDown(KeyCode.RightArrow)) { // Nhấn phím mũi tên phải
            action = 2;
            animator.SetInteger("Action", action);
        }

        if (Input.GetKeyUp(KeyCode.RightArrow)) { // Nhả phím mũi tên phải
            action = 0;
            animator.SetInteger("Action", action);
        }

    }
}
```

Hình 5.22. Viết mã lệnh điều khiển với phím mũi tên trái, phải

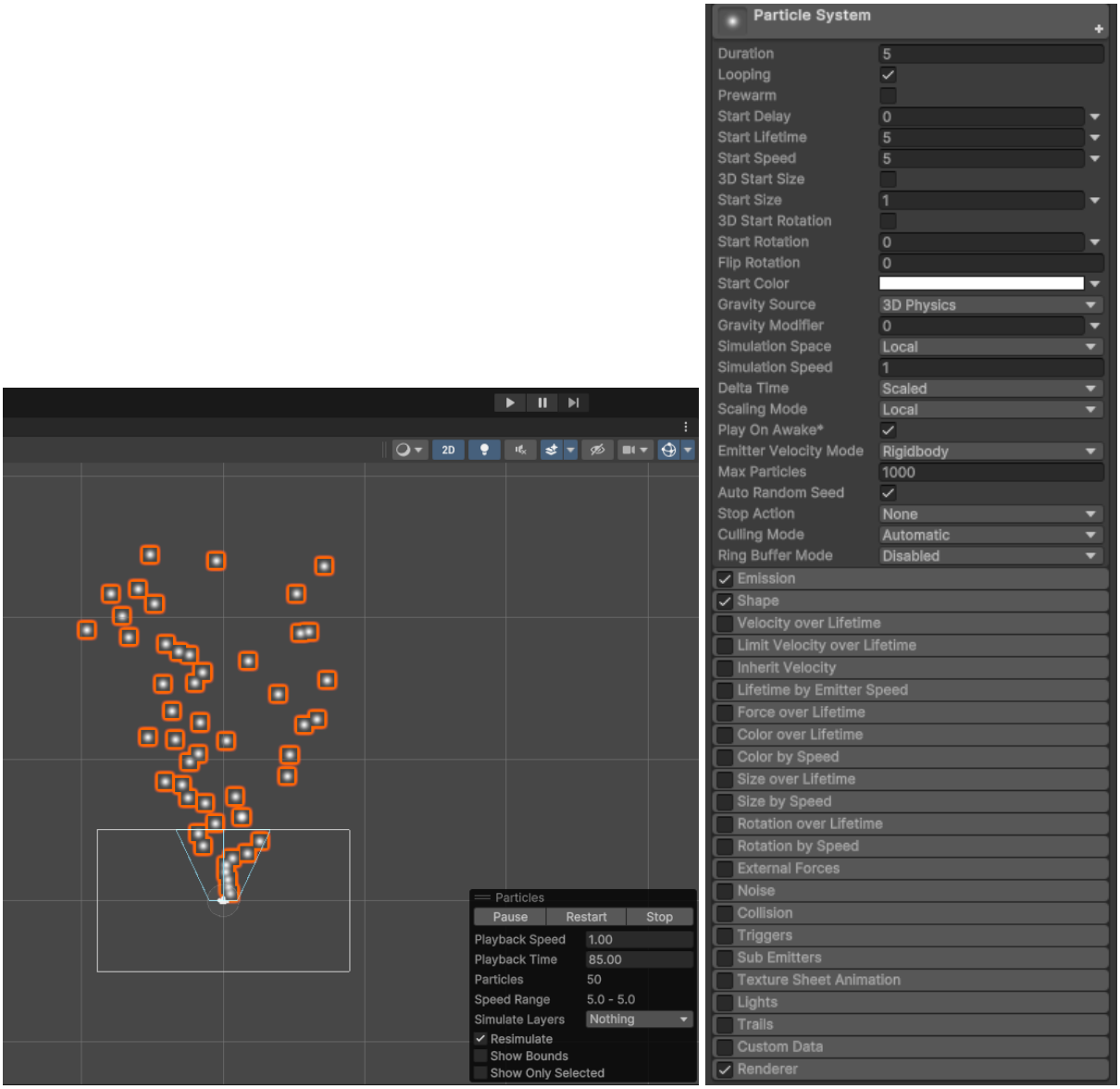
Trong mã lệnh trên, đầu tiên ta cần lấy được Animator của Player, mỗi lần người dùng nhấn phím thì biến action sẽ được gán một giá trị, giá trị đó sẽ được gửi qua cho biến Action bên máy trạng thái. Máy trạng thái sẽ dựa vào giá trị của Action để chuyển trạng thái phù hợp.

Ngoài biến với giá trị kiểu Int, Unity còn cho phép thiết lập biến với các giá trị kiểu số thực, biến kiểu bool hoặc biến kiểu Trigger. Tùy thuộc vào bài toán cụ thể mà có thể dùng các biến tương ứng. Trong mũi tên chỉ trạng thái, phần Inspector, Unity cũng cho phép thiết lập các phép so sánh như Greater, Less, Note Equal… Tùy vào yêu cầu của biến mà có thể chọn phép so sánh phù hợp.

7. Hiệu ứng hạt trong Unity

Unity hỗ trợ tạo các hiệu ứng sinh động như khói, nước, lửa, pháo hoa, vụ nổ… bằng một nhóm đối tượng gọi là Effects. Nhóm đối tượng này bao gồm Particle System (hệ thống hạt), Trail (tạo vệt đuôi), và Line (tạo đường thẳng). Phần này sẽ giới thiệu một số chức năng cơ bản của hệ thống hạt (Particle System).

Particle System được thêm vào bằng cách trên Hierarchy, chọn Game Object, chọn Effects, chọn Particle System. Khi tạo xong Unity sẽ tạo ra một đối tượng hiệu ứng trên Scene và một bộ các thuộc tính như trong Hình 5.23.



Hình 5.23. Hiệu ứng hạt trên Scene và các thuộc tính liên quan

Do hiệu ứng hạt là một hệ thống khá phức tạp trong Unity nên trong giáo trình này chỉ đề cập tới một số mục cơ bản như sau:

- Trên màn hình Scene, phần ***Particles*** có các nút để điều khiển như Pause, Restart, Stop. Các nút này dùng để điều khiển hiệu ứng hoạt động hay tạm ngưng.
- Cụm thuộc tính chính (**Particle System**): Đây là cụm thuộc tính quy định một số tính chất tổng quát của hệ thống hạt, như chế độ lặp lại, khoảng thời gian phát hạt hay gắn trọng lực cho hạt… Các thuộc tính chính được trình bày như trong Bảng 5.3

Bảng 5.3. Các thuộc tính chính của Particle System

Thuộc tính	**Chức năng**
Duration	Thời gian chạy hiệu ứng, tính bằng giây, nếu không thiết lập chế độ lặp lại thì sau khoảng thời gian này, hiệu ứng sẽ tắt
Looping	Cho phép hiệu ứng lặp lại hoặc không
Start Delay	Thời gian trễ, thời gian bị trễ khi bắt đầu hiệu ứng
Start Lifetime	Thời gian để bắt đầu lại vòng đời của hạt
Start Speed	Tốc độ ban đầu của hạt
3D Start Size, 3D Start Rotation, Start Rotation, Flip Rotation	Kích thước của hạt, nếu được chọn sẽ cho thiết lập kích thước của hạt theo x, y,z. Tương tự với phép quay và phép lật cho hạt
Start Color	Màu ban đầu của hạt. Lưu ý, Unity cho phép thiết lập màu Gradient để pha trộn các màu giúp hiệu ứng đẹp hơn
Gravity Source, Gravity Modifier	Cho phép thiết lập trọng lực cho hạt, nếu được thiết lập hạt khi bay ra khỏi hệ thống sẽ rơi xuống.
Simulation Space, Simulation Speed	Giúp hệ thống minh họa không gian cho hạt, không gian dựa trên trục tọa độ, còn tốc độ thì giúp hạt bay nhanh hoặc chậm tùy vào tham số thiết lập
Delta Time, Scale Mode	Chế độ của Time khi hiệu ứng ở trạng thái tạm dừng (Pause)
Play on Awake	Cho phép hiệu ứng hoạt động ngay khi Game bắt đầu
Emitter Velocity Mode	Chế độ vận tốc của các hạt khi phát ra
Max Particles	Số lượng hạt tối đa
Auto Random Seed	Chế độ cho phép thực hiện ngẫu nhiên sự phát hạt
Stop Action	Chế độ khi hiệu ứng ngưng hoạt động
Culling Mode	Chế độ chọn lọc
Ring Buffer Mode	Chế độ quản lý bộ đệm trong sự phát hạt

- ***Emission***: Thuộc tính này quản lý chế độ phát ra của hạt. Các thuộc tính đi kèm như *Rate over Time*: cho phép quản lý tỉ lệ phát hạt trong một khoảng thời gian nhất định; Rate over Distance: cho phép quản lý tỉ lệ phát hạt trong một khoảng cách nhất định; Sự phát hạt còn được quản lý bởi một mảng các đối tượng Bursts(được quản lý bởi thời gian, số lượng, số vòng quay, thời gian sống và xác suất).
- ***Shape***: Quản lý hình dạng của hạt phát ra. Có thể chọn các loại hình dạng trong tham số Shape như Cone, Sphere, Donut, Box, Sprite… Tùy vào hình dạng do người dùng chọn mà sẽ có các thông số đi kèm như bán kính, chiều cao, tọa độ…
- ***Renderer:*** Thuộc tính này cho phép quản lý sự hiển thị của hạt khi phát ra, một số tham số đi kèm như Material hoặc Mesh cho phép quản lý sự hiển thị của hạt khi phát ra được hiệu quả hơn.
- ***Texture Sheet Animation***: Cho phép xây dựng các hạt phát ra là các hình ảnh.
- ***Sub Emitters:*** *Quản lý hạt con, là loại hạt phát ra từ hạt gốc, kiểu như pháo hoa khi nổ hoặc sự bắn của nước khi rơi xuống*
  Và một số thuộc tính khác.

8. Kết chương

Chương này giới thiệu cách tạo chuyển động trong game bằng Animation. Đây là thành phần giúp cho các đối tượng game có thể tạo nên hiệu ứng chuyển động thông qua việc thay đổi các tham số như vị trí, co dãn hoặc thay đổi ngoại hình cho nhân vật. Qua việc sử dụng hiệu ứng, chương này cũng giới thiệu cách quản lý các chuyển động và viết lệnh để điều khiển các chuyển động này. Chương này đồng thời cũng giới thiệu về hệ thống hạt, một hệ thống giúp tạo ra các hiệu ứng linh hoạt trên game.

Bài tập

1. Animation, Animation Controller và Animator giống và khác nhau như thế nào.

2. Hãy tạo một chuyển động như sau: Từ hình tròn chuyển qua hình vuông, rồi chuyển qua hình tam giác và ngược lại trong thời gian 2s.

3. Hãy tạo một chuyển động biến đổi màu sắc cho 1 đối tượng chuyển từ màu vàng qua màu đỏ và qua màu xanh trong thời gian 1.5s.

4. Giải thích vai trò của các thuộc tính của Animator

5. Hãy tạo trạng thái Idle là một vật có 1 cái đầu với 2 mắt đổi màu và miệng mấp máy.

6. Thực hành tạo các trạng thái Idle, RightRotate, LeftRotate như trong Hình 5.18.

7. Hãy tạo các biến tương ứng để điều khiển các lệnh trong bài tập 6.

8. Hiệu ứng hạt là thành phần hay đối tượng game? Cho ví dụ và giải thích

9. Giải thích các tham số cơ bản của Particle System

10. Tìm tài liệu và thực hiện hệ thống pháo hoa với Particle System

Chương 6. Xây dựng thế giới nhân vật

Chương này hướng dẫn người đọc tìm hiểu về cách xây dựng nhân vật 2D trong Unity. Từ một hình ảnh ban đầu khi đưa vào Unity, hình sẽ được chuyển về dạng Sprite, từ đó tạo nên hình dáng và các cử động của nhân vật. Bên cạnh đó, chướng này cũng hướng dẫn người đọc cách tạo và quản lý bản đồ trong game để nhân vật hoạt động.

1. Thế giới nhân vật trong Game 2D

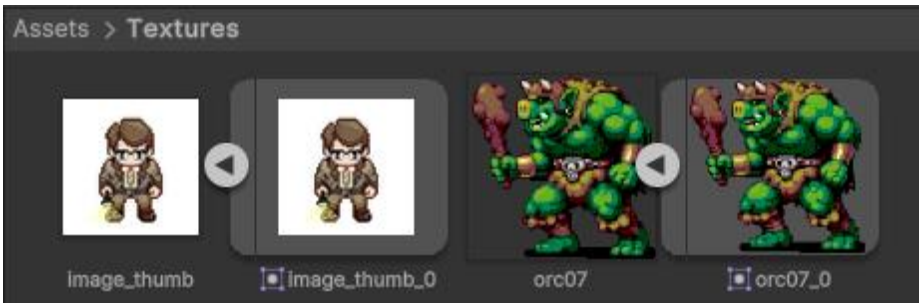
Trong lập trình Game nói chung và lập trình Game 2D nói riêng, việc xây dựng thế giới nhân vật đóng một vai trò quan trọng. Thế giới nhân vật bao gồm cách xây dựng bản đồ, quản lý bản đồ, xây dựng đường đi và chướng ngại vật, đồng thời với đó là xây dựng các hoạt động trong cơ thể nhân vật dưới dạng Sprite.

Hình 6.1. Một mô hình thế giới nhân vật trong Game FlyWorld (nguồn: github.com/Fy-/FyWorld)

Unity hỗ trợ xây dựng thế giới bằng Tilemaps và Tile Palettes, đây là hai công cụ dùng để tạo bản đồ, đường đi và tổ chức các màn trong Game. Ngoài ra, để nhân vật có sự sinh động, Unity hỗ trợ thiết kế và xử lý Sprite.

2. Sprite trong Unity

Sprite là các đối tượng đồ họa 2D sử dụng cho các nhân vật, đối thủ, đạn, đạo cụ và các thành phần khác của một trò chơi 2D. Sprite được tạo ra bằng cách biến đổi từ Texture (ảnh png, jpg, bmp…) thành Sprite với thuộc tính Texture Type. Sprite có hai loại là Single Sprite và Multiple Sprite, Single Sprite là loại Sprite được tạo ra bằng một Texture (Hình 6.2), trong khi đó Multiple Sprite được tạo ra bằng cách cắt một số phần trong một bức ảnh bằng công cụ Sprite Editor (Hình 6.3).



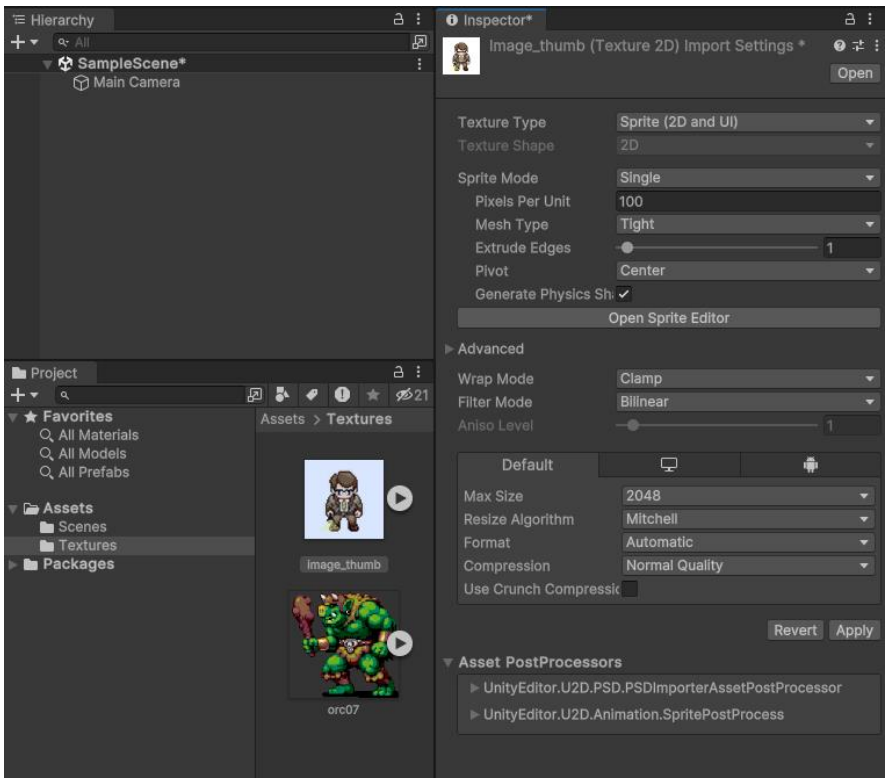
Hình 6.2. Mỗi Texture gốc sẽ cho ra một Single Sprite

Hình 6.3. Sprite bên phải được tạo ra từ các phần của một Texture bên trái

Để tạo được Sprite, Unity có hỗ trợ các loại công cụ như sau:

- ***Sprite Creator***: Công cụ này cho phép tạo ra Sprite trực tiếp trên Unity mà không cần dùng bên thứ ba.
- ***Sprite Editor***: Công cụ cho phép trích xuất các thành phần trong một Texture để tạo nên một Multiple Sprite. Ví dụ như trong Hình 6.1, từ Texture bên trái ta có thể tạo ra một Sprite như hình bên phải.
- **Sprite Renderer**: Là một thành phần để hiển thị hình ảnh như là một Sprite trong 2D và 3D.
- **Sprite Packer**: Công cụ này cho phép tùy chọn và sử dụng hiệu năng của bộ nhớ video trong dự án.

Để sử dụng Sprite trong dự án 2D, trước hết ta cần đưa hình ảnh vào trong dự án, sau đó trên phần Inspector, trong thuộc tính Texture Type chọn Sprite (2D and UI) và trong thuộc tính Sprite Mode chọn Single nếu như muốn tạo một Single Sprite (Hình 6.4). Một số thuộc tính chính của Sprite được mô tả như trong Bảng 6.1.



Hình 6.2. Tạo Sprite bằng cách thay đổi thuộc tính Texture Type

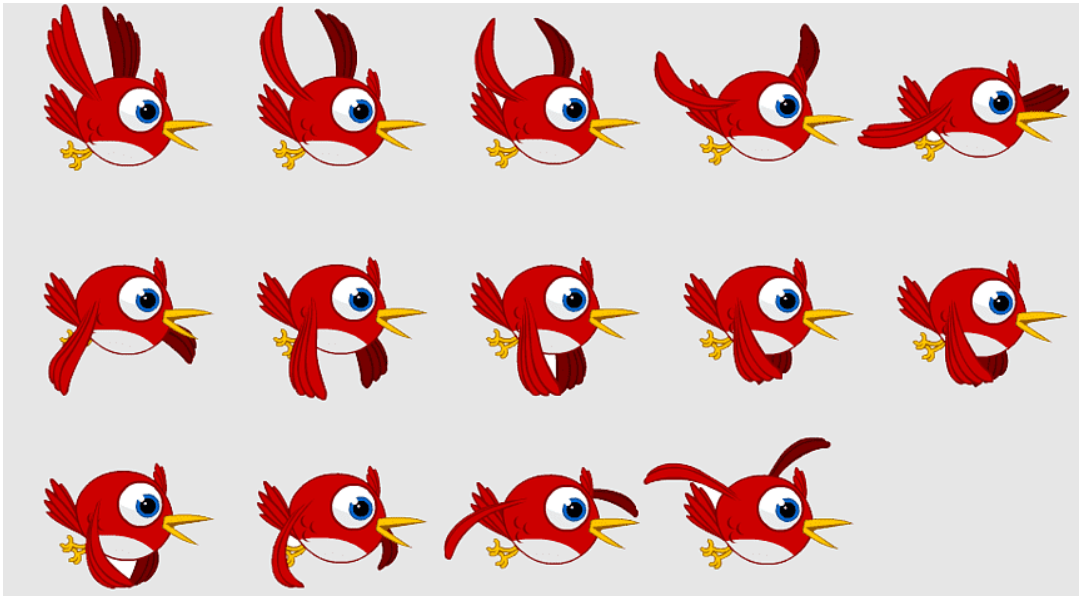
Bảng 6.1. Các thuộc tính chính của một Texture khi chuyển qua Sprite

Thuộc tính	**Chức năng**
Texture Type	Quy định về các kiểu Texture, nếu muốn xử lý Sprite, cần chọn giá trị Sprite (2D and UI)
Sprite Mode	Có 3 giá trị là Single, Multiple và Polygon. *Single* cho phép thiết lập Sprite đơn, nghĩa là một Texture chỉ sinh ra một Sprite; *Multiple* cho phép tạo một Sprite từ nhiều thành phần trong Texture bằng Sprite Editor; *Polygon* cho phép cắt Sprite theo hình đa giác trong Sprite Editor.
Pixels Per Unit	Số lượng Pixel trên một đơn vị ô vuông. Số càng lớn thì Sprite hiện lên trên Scene càng nhỏ.
Mesh Type	Định nghĩa kiểu lưới cho Sprite: *Full Rect* là lưới hình chữ nhật bằng với kích thước của Sprite; *Tight* là lưới được tính dựa theo giá trị alpha trong hệ màu, mặc định sẽ chọn giá trị *Tight*
Extrude Edges	Thuộc tính cho phép thổi phồng cạnh lên trong một số trường hợp
Các thuộc tính khác	Unity hỗ trợ thêm một số thuộc tính khác như Advance (các thuộc tính nâng cao), Wrap Mode (chế độ căn chỉnh), Max Size (kích thước tối đa)…

Sau khi chuyển từ Texure về Sprite (2D and UI), người dùng có thể kéo ra màn hình, lúc này Sprite chính là một đối tượng Game. Ta có thể thêm các thành phần như một đối tượng Game thông thường.

3. Công cụ Sprite Editor

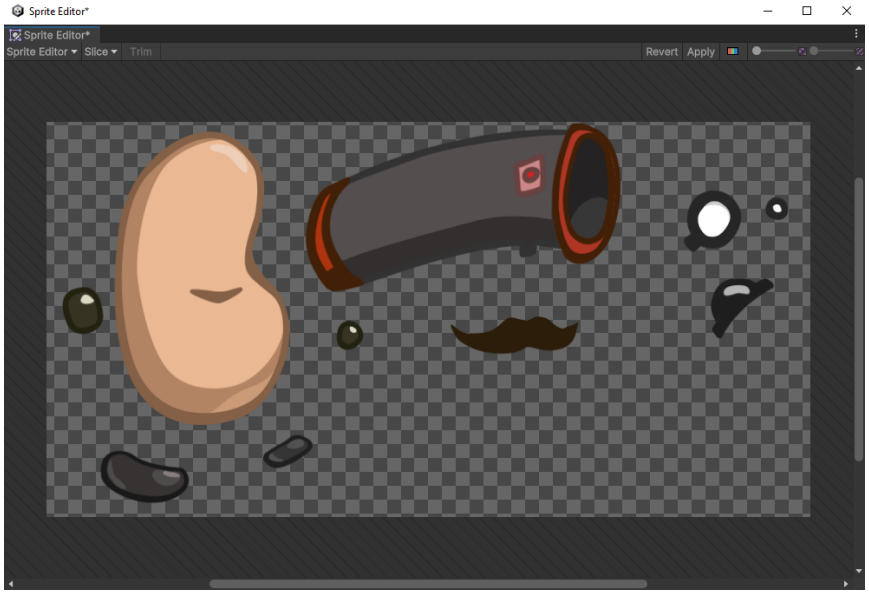
Một Texture chứa các Sprite có thể ở hai dạng là dạng thành phần rời rạc, nghĩa là các thành phần trong Sprite được thiết kế trên một ảnh, hoặc ở dạng lưới, nghĩa là các hình ảnh chuyển động của Sprite được bố trí theo một lưới ngang hoặc dọc. Hình 6.3 bên trái là một Texture chứa các thành phần Sprite rời rạc, trong khi đó Hình 6.5 là một Texture chứa các thành phần dạng lưới.



Hình 6.5. Texture chứa các Sprite dạng lưới (nguồn: www.anyrgb.com)

Công cụ Sprite Editor được sử dụng để chuyển đổi một Texture chứa các thành phần của Sprite rời rạc hoặc lưới các Sprite thành các Sprite riêng biệt dùng trong đối tượng Game.

Để sử dụng được Sprite Editor, trước hết chọn Texture, trong phần Inspector, thiết lập Sprite Mode là Multiple, sau đó click chuột vào Sprite Editor để mở Sprite Editor (Hình 6.6), Sprite Editor cũng có thể mở ra bằng cách trên thanh Menu, chọn Window, chọn Sprite Editor.

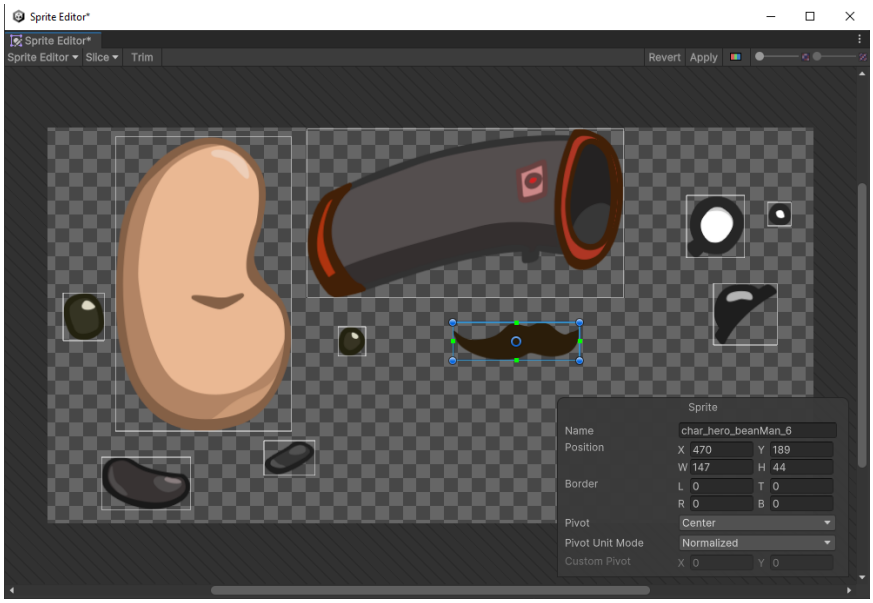


Hình 6.6. Công cụ Sprite Editor

Lúc này cửa sổ Sprite Editor bao gồm các chức năng như sau:

- Ở góc trên, bên trái là chức năng chọn các chế độ Sprite và chức năng cắt các Sprite ra thành các phần riêng biệt. Bên cạnh đó có nút Trim, dùng để loại bỏ các khoảng trống dư thừa trong ảnh png cho các Sprite
- Ở góc trên, bên phải có các chức năng như Revert (đảo ngược kết quả), Apply (áp dụng các tham số để cắt) và một số chức năng như màu sắc, thu phóng hình ảnh.

Dùng chuột nhấn, kéo và rê chuột để vẽ các hình chữ nhật xung quanh các thành phần của Sprite, nhấn nút Trim để loại bỏ các khoảng trống dư thừa, lúc này Sprite Editor sẽ vẽ các khung hình chữ nhật tối ưu bao quanh. Bấm chuột vào một Sprite bất kỳ, ta có các tham số như trong Hình 6.7.



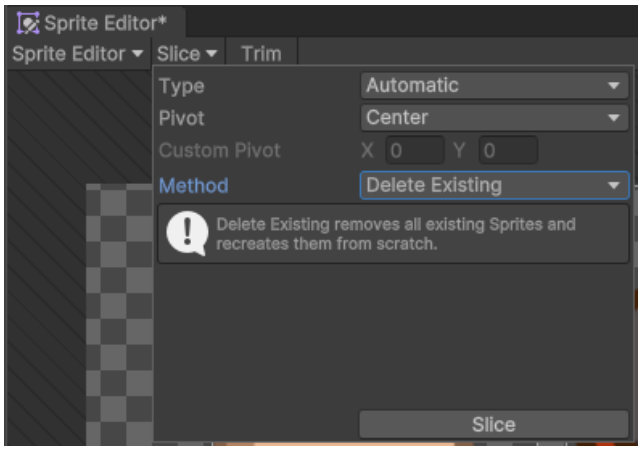
Hình 6.7. Các Sprite và tham số liên quan đến 1 Sprite

Các tham số này bao gồm:

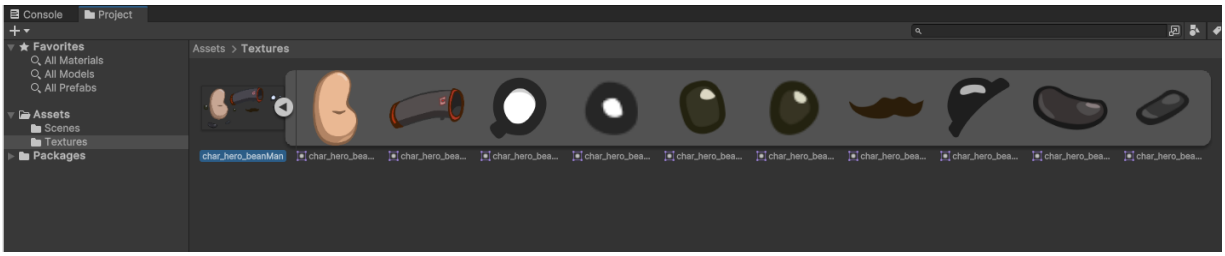
- *Name*: Là tên của Sprite, tên này do hệ thống tự đặt, có đánh số thứ tự, để dễ quản lý sau này, ta có thể đổi tên cho nó về dạng tiền tố kiểu như idle_... hoặc leftWalk_...
- *Position*: Là vị trí của Sprite trong Texture, ví trí bao gồm tọa độ X, Y và chiều rộng, chiều cao của Sprite.
- *Border*: Là đường viền của Sprite, bao gồm các thông tin như top, bottom, left, right. Đường viền này chỉ quan trọng khi Spite được cắt dưới dạng lưới.
- *Pivot*: là điểm tựa để xoay, mặc định điểm xoay là Center. Unity cung cấp các loại điểm xoay khác như TopLeft, TopRight….
- *Pivot Unit Mode*: Là chế độ xác định điểm xoay.

Các tham số này thường được Unity thiết lập mặc định dựa trên việc xác định tọa độ của các điểm ảnh. Người dùng có thể tùy chỉnh các tham số để cho ra các Sprite phù hợp.

Tiếp theo, ở góc trên, bên trái, bấm chuột vào Slice, chọn Type là Automatic, click chuột vào nút Slice (Hình 6.8), sau đó nhấn nút Apply, tắt Sprite Editor để cắt thành các Sprite như Hình 6.9. Ngoài Automatic, Unity còn cung cấp các loại cắt dạng lưới, sẽ được đề cập ở phần sau.



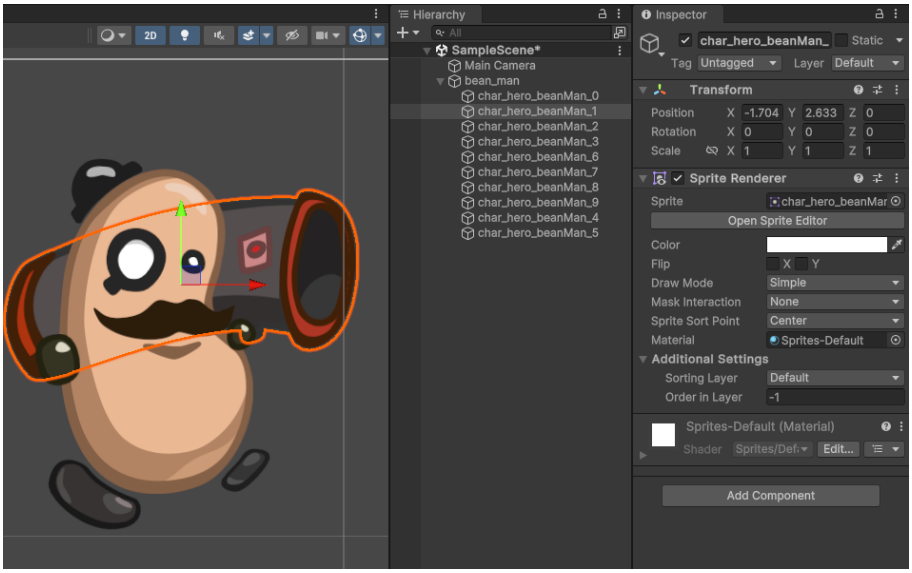
Hình 6.8. Chức năng Slice dùng để cắt Texture thành các Sprite



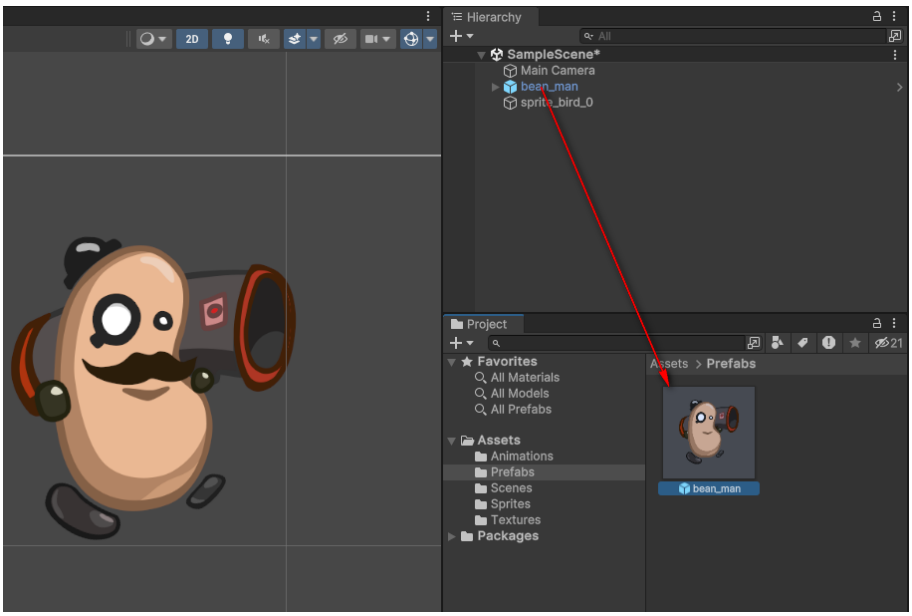
Hình 6.9. Texture được cắt thành các Sprite

Để xây dựng một nhân vật từ các Sprite sau khi cắt, chúng ta chỉ cần kéo các thành phần ra giao diện Scene, sau đó thiết lập các thành phần vào đúng vị trí. Có thể phóng to, thu nhỏ hoặc xoay các đối tượng cho phù hợp. Để thiết lập vị trí của các Sprite phù hợp, trong thành phần Sprite Renderer, đặt giá trị cho thuộc tính Order In Layer phù hợp, số càng lớn thì càng ở phía trước.

Để dễ quản lý, ta có thể tạo ra một Empty Object, sau đó kéo hết các thành phần này vào Empty Object vừa tạo (Hình 6.10). Lưu ý là có thể thiết lập các thông số khác cho Empty object này như RigidBody, Collider. Khi thực hiện xong, chuyển nhân vật xuống thành Prefab để dùng cho lần sau (Hình 6.11)



Hình 6.10. Tạo nhân vật từ các Sprite

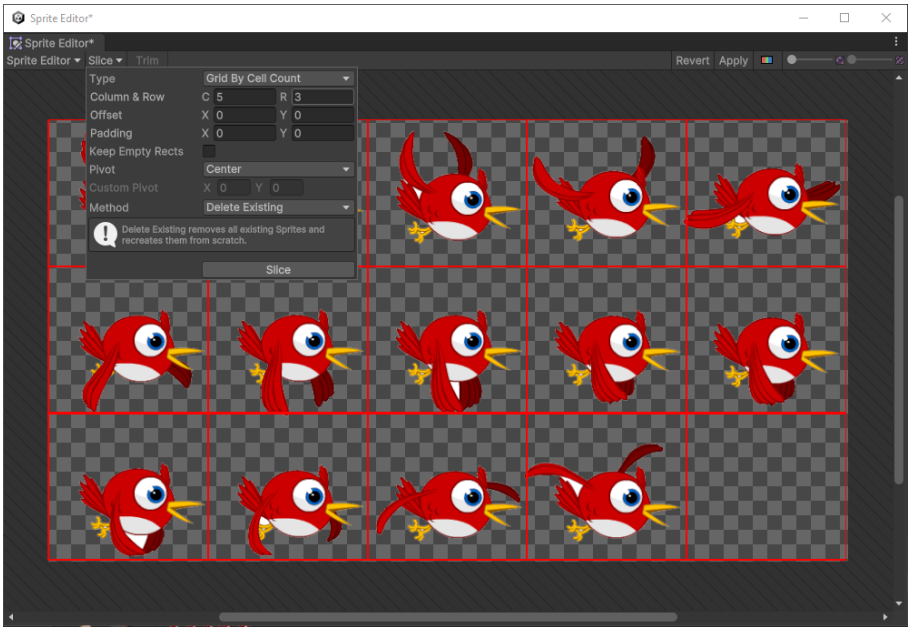


Hình 6.11. Chuyển thành Prefab

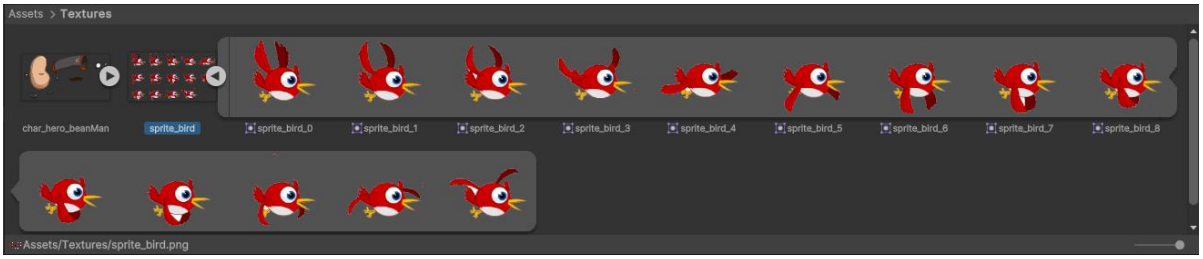
Để xây dựng các Sprite theo dạng lưới, ta cần có Texture được thiết kế các nhân vật đều nhau theo hàng và cột (Hình 6.5), trong phần Inspector, chọn Sprite Mode là Multiple. Mở cửa sổ Sprite Editor bằng cách nhấn Open Sprite Editor, chọn mục Slice, trong thuộc tính Type, Unity cung cấp 3 loại lưới cơ bản là:

- *Grid By Cell Size*: Phân chia lưới theo kích thước, khi đó các ô được chia theo kích thước thiết lập trong Pixel Size của X và Y. Tham số *Offset* quy định độ dày của đường viền còn tham số *Padding* quy định khoảng cách giữa các ô;
- *Grid By Cell Count*: Phân chia lưới theo số lượng ô, lúc này các Sprite sẽ được cắt theo số lượng ngang và dọc được thiết lập. Tham số Offset và Padding tương tự như trên.
- *Isometric Grid*: Phân chia lưới dạng Isometric với kích thước và Offset cho trước.

Hình 6.12 cho thấy một Texture dạng lưới được cắt theo *Grid By Cell Count* với số cột bằng 5 và số dòng bằng 3. Nhấn Slice để cắt thành các Sprite, nhấn nút Apply để áp dụng, ta có kết quả như trên Hình 6.13.



Hình 6.12. Tạo Texture dạng lưới với số cột và số dòng



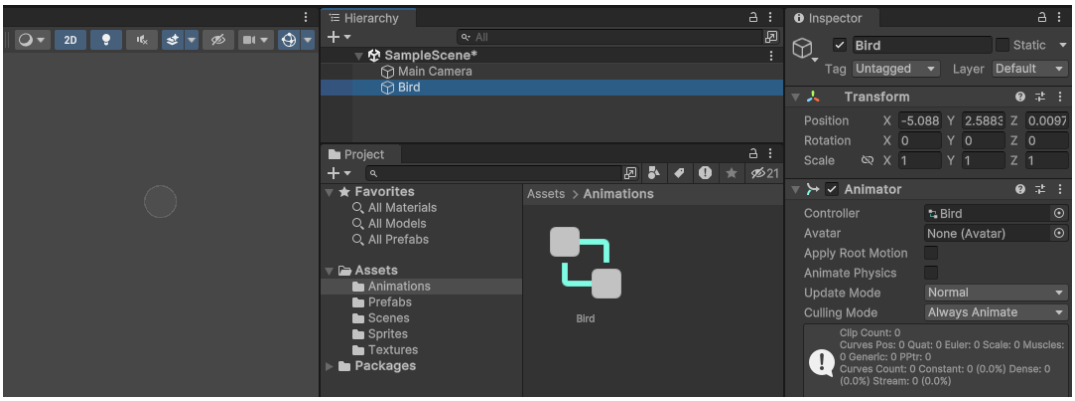
Hình 6.13. Các Sprite được tạo sau khi cắt hình dạng lưới

Sau khi có các Sprite, ta có thể tạo các ảnh động cho nhân vật bằng chức năng Animation.

4. Xây dựng Animation với Sprites

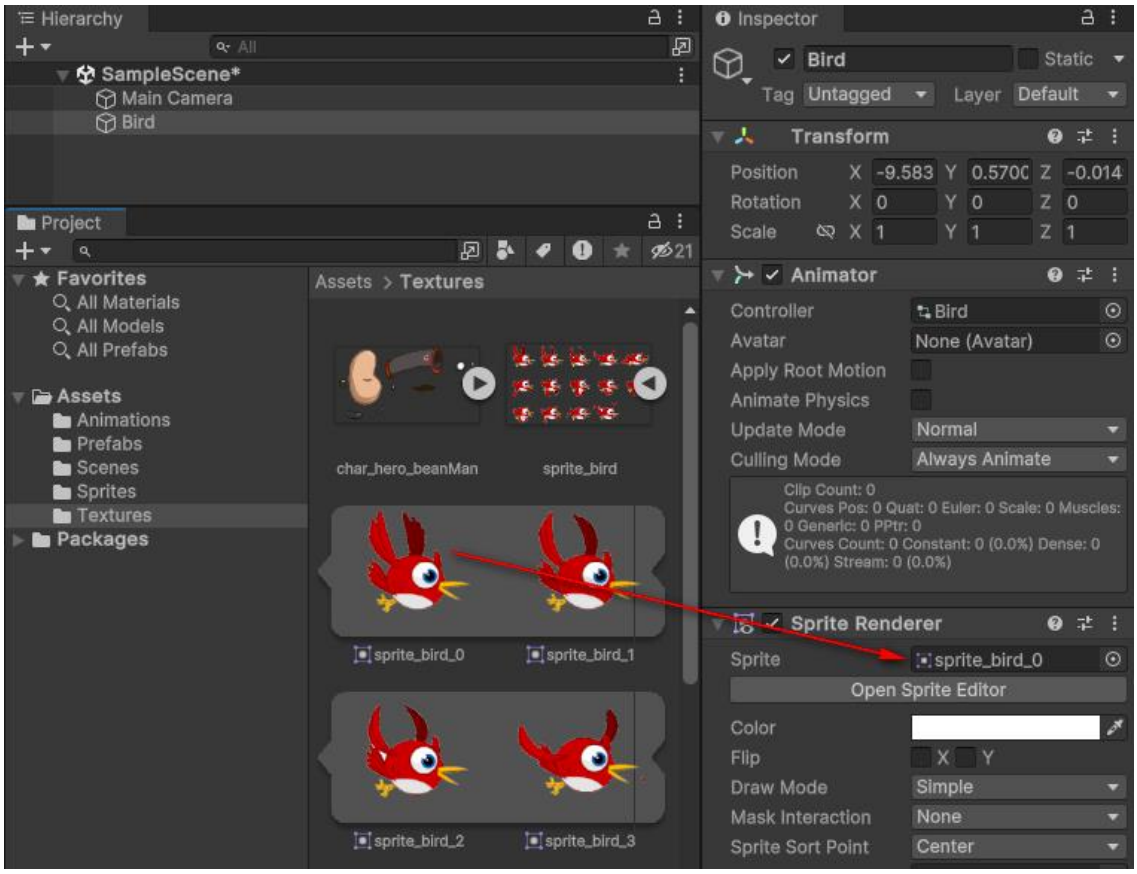
Để xây dựng Animation với các Sprites đã thiết kế, người ta sử dụng chức năng Animation để chuyển qua lại các sprite giữa các frames. Các bước sau đây sẽ thực hiện tại Animation cho các Sprite:

Bước 1: Tạo một đối tượng Empty đặt tên là Bird, trong Inspector thêm vào thành phần Animator. Trong thư mục Animations, click phải, chọn Create, chọn Animation Controller, đặt tên là Bird, sau đó kéo vào phần Controller như Hình 6.14.



Hình 6.14. Tạo một Empty Object, gắn Animator với Animation Controller

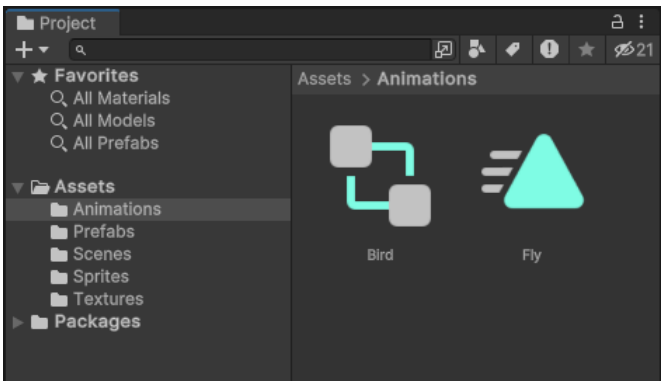
Bước 2: Trong phần Inspector, chọn Add Component, thêm vào thành phần Sprite Renderer. Trong thuộc tính Sprite, bấm tìm và chọn lấy Sprite đầu tiên của trạng thái (sprite_bird_0 hoặc sprite khác nếu tạo các Clip khác). Hoặc có thể tìm tới Sprite đầu tiên rồi nhấp chuột và kéo vào thuộc tính Sprite như trong Hình 6.15.



Hình 6.15. Gán Sprite đầu tiên cho Sprite Renderer

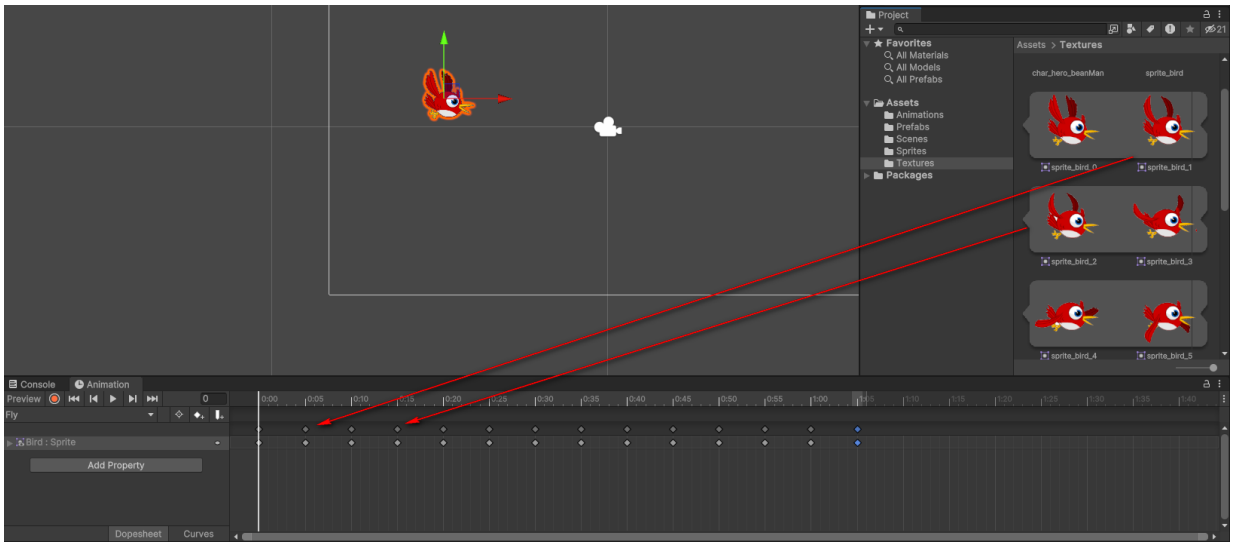
Bước 3: Tạo các Clip, thường thì một nhân vật sẽ có nhiều Clip, ví dụ như Idle, LeftRunning, RightRunning, … Mỗi một Clip sẽ có các Frame tương ứng trong Sprite, khi dùng Sprite Editor để cắt, cần ghi nhớ các Frame này. Trong ví dụ này, chỉ tạo một Clip là Fly, vì Sprite này chỉ có một động tác bay.

Trước hết, vào Window, chọn Animation, Animation để mở cửa sổ Animation, sau đó kéo về vị trí phù hợp. Chọn Bird trong Hierarchy, bấm chọn Create trong thẻ Animation, đặt tên Clip là Fly và lưu vào thư mục Animations (Hình 6.16).



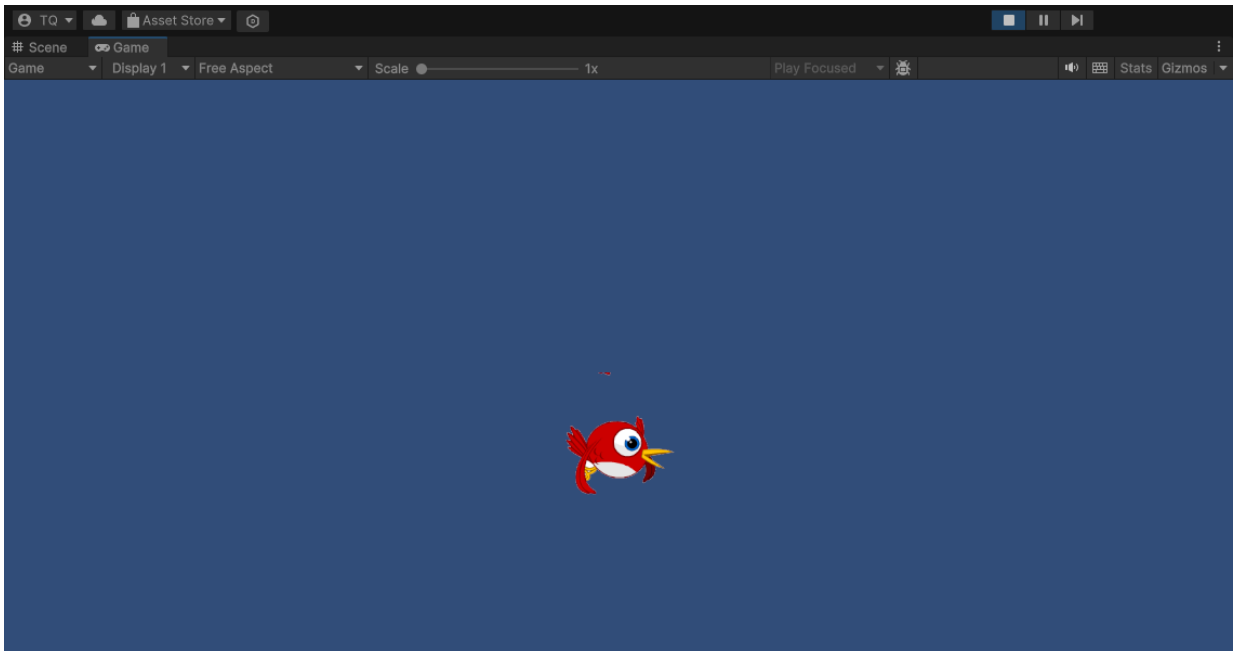
Hình 6.16. Tạo một Animation Fly

Bước 4: Trong thẻ Animation, chọn Add Property, chọn Sprite Renderer, chọn Sprite và bấm dấu + để thêm Animation cho Clip này. Chọn hết các Sprites (trừ sprite_bird_0), sau đó thẻ vào phần Frame của Animation, thay đổi vị trí của các Frame cho mỗi 0:05s cho một Frame (Hình 6.17).



Hình 6.17. Gán các sprite cho mỗi 0:05s

Bước 5: Nhấn nút Play để thấy kết quả. Có thể kéo nhân vật này vào thành Prefab để sử dụng về sau.



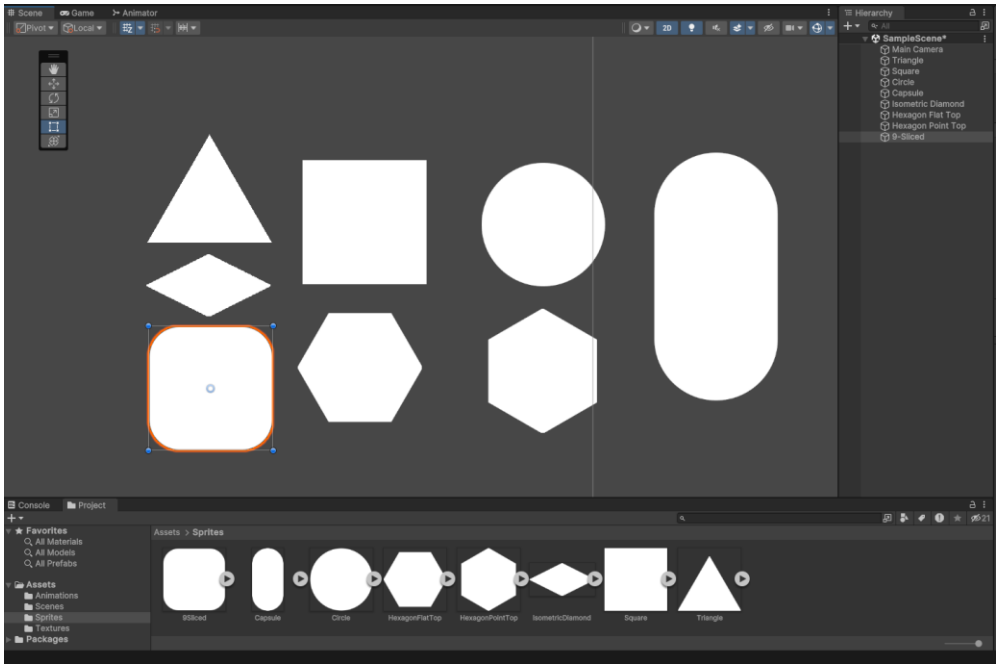
Hình 6.18. Hình ảnh khi chạy

Nếu có nhiều Clip, có thể thao tác tương tự cho từng Clip để tạo ra các Clip, sau đó tiến hành điều khiển các Clip này như đã trình bày ở Chương 5.

5. Công cụ Sprite Creator

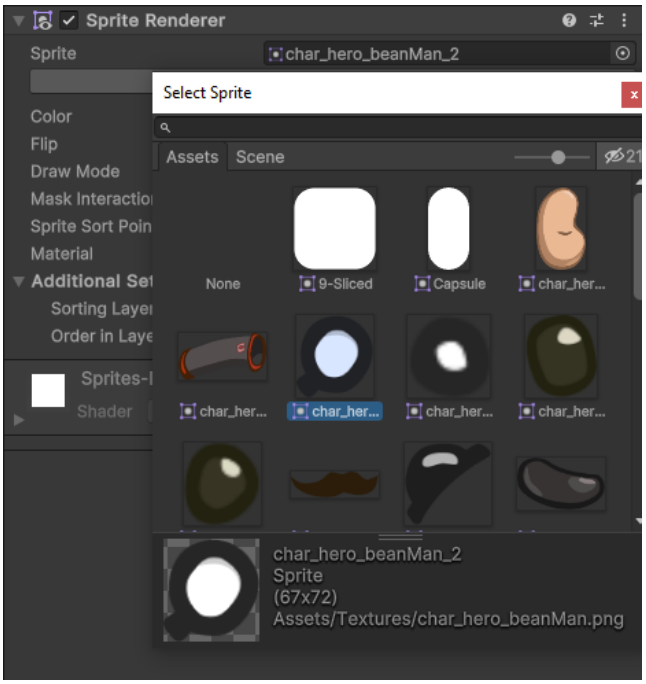
Công cụ này cho phép tạo ra một Sprite 2D thủ công, người dùng có thể thay thế bằng hình ảnh hoặc một đối tượng khác sau này. Để tạo ra một Sprite, trong Project, click phải, chọn Create, chọn 2D, chọn Sprites và lựa chọn các loại Sprite mà bạn muốn.

Unity cung cấp các loại Sprite như *Triangle* (tam giác), *Square* (hình vuông), Circle (hình tròn), Capsule (hình con nhộng), Isometric diamond (hình kim cương), Hexagon flat-top (hình lục giác tù), hexagon pointed-top (hình lục giác nhọn), 9-sliced (hình 9 cạnh). Sau khi đã tạo xong trong thư mục tại Project, lúc này có thể kéo Sprite lên màn hình Scene để sử dụng như một Sprite bình thường (Hình 6.19).



Hình 6.19. Các loại Sprite được tạo thủ công

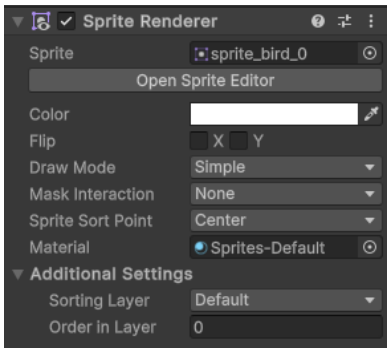
Để thay thế hình dạng, màu sắc của Sprite đã tạo, trên Inspector, trong thành phần Sprite Renderer, click chọn Sprite để chọn Sprite tương ứng (Hình 6.20). Chúng ta cũng có thể đổi màu sắc cho Sprite bằng cách chọn mục Color.



Hình 6.20. Lựa chọn Sprite để thay thế Sprite thủ công

6. Sprite Renderer và Sprite Mask

Công cụ Sprite Renderer cho phép quản lý việc hiển thị Sprite lên một Scene, công cụ này thường được gắn vào một Empty Object hoặc gắn vào một Texture để biến Empty Object hoặc Texture đó thành Sprite (Hình 6.21)



Hình 6.21. Công cụ Sprite Renderer

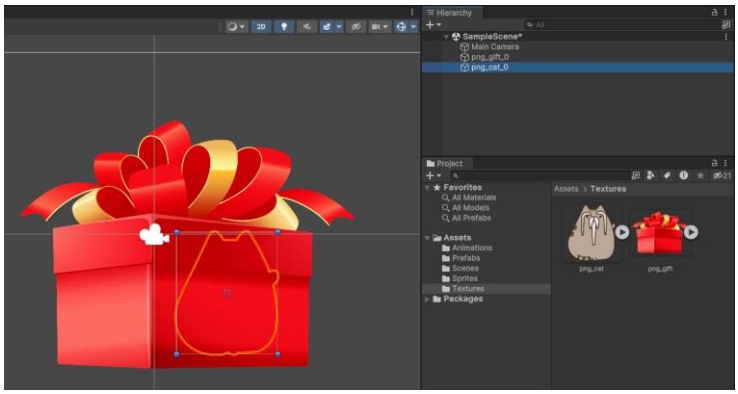
Công cụ Sprite Renderer có các thuộc tính được mô tả như trong Bảng 6.2.

Bảng 6.2. Một số thuộc tính của Sprite Renderer

Thuộc tính	**Chức năng**
Sprite	Định nghĩa một ảnh để hiển thị
Color	Định nghĩa màu sắc cho Sprite, màu này sẽ pha trộn với màu của ảnh
Flip	Lật ảnh theo trục X hoặc trục Y
Material	Định nghĩa loại chất liệu để kiến tạo ảnh
Draw Mode	Thiết lập sự co dãn của Sprite khi thay đổi chiều của nó. Thuộc tính này có các giá trị như sau: • *Simple*: Giữ kích thước mặc định; • *Sliced*: Lựa chọn chế độ này nếu Sprite là loại 9 cạnh, trong đó cho phép thiết lập kích thước cạnh; • *Continuous*: Đây là chế độ mặc định của Tile Mode (sẽ được đề cập trong mục 6); • *Adaptive*: Ở chế độ này Sprite sẽ co giãn ở mức phù hợp, nếu giá trị co dãn vượt quá *Stretch Value* thì sẽ bị cắt; • *Stretch Value*: Giá trị từ 0 đến 1 để thiết lập độ co giãn cho Sprite
Sorting Layer	Thuộc tính này cho phép thiết lập thứ tự ưu tiên trong quá trình kiến tạo ảnh. Thứ tự ưu tiên sẽ ảnh hưởng đến việc che khuất của ảnh trong màn chơi (Hình 6.22)
Order In Layer	Thiết lập giá trị ưu tiên trong *Sorting Layer*, giá trị thấp nhất sẽ được kiến tạo đầu tiên.
Mask Interaction	Thiết lập chế độ giao thoa giữa các Sprite. Một số giá trị sẵn có như sau: • *None*: Không thiết lập chế độ giao thoa • *Visible Inside Mask*: Sprite kiến tạo khi có một Sprite Mask phủ lên nó, nhưng không hiển thị bên ngoài nó. • *Visible Outside Mask*: Sprite có thể nhìn thấy bên ngoài Sprite Mask nhưng không hiển thị bên trong nó. Để hiểu hơn về thuộc tính này có thể đọc thêm về Sprite Mask được trình bày trong phần tiếp theo của mục này
Sprite Sort Point	Thuộc tính cho phép thiết lập và tính toán tọa độ giữa Sprite và Camera.

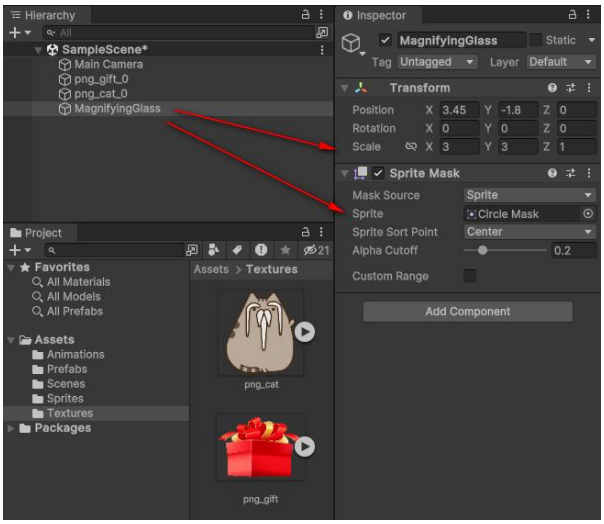
Công cụ Sprite Mask là công cụ cho phép thiết lập mặt nạ giữa các Sprite với nhau, nghĩa là cho phép một Sprite là mặt nạ thì có thể "soi" các Sprite khác. Để minh họa về Sprite Mask, hãy xem ví dụ như sau:

Đầu tiên, tìm hai hình bằng png, một hình là gói quà, một hình là vật nằm bên trong gói quà, đưa hai hình vào Unity, kéo ra màn hình Scene sao cho vật bị khuất bên trong gói quà như Hình 6.22.



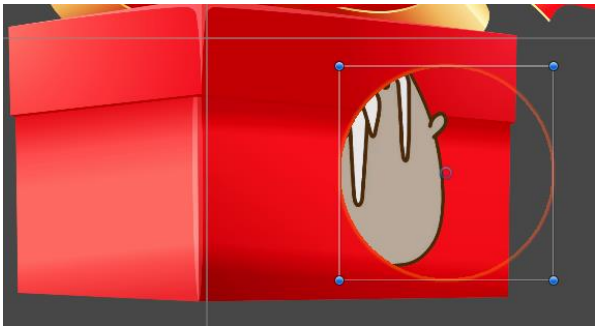
Hình 6.22. Hai đối tượng Sprite

Tiếp theo, tạo mới một Sprite Mask bằng cách click phải lên Hierarchy, chọn 2D Object, chọn Sprite Mask, đặt tên là Magnifying Glass (kính lúp), thiết lập kích thước (Scale) và hình dạng (Sprite) phù hợp như trong Hình 6.23.



Hình 6.23. Tạo mới Sprite Mask

Chọn lại đối tượng png_cat_0, thay đổi giá trị lựa chọn trong tham số Mask Interaction, di chuyển kính lúp để xem kết quả (Hình 6.24).



Hình 6.24. Kết quả khi thay đổi tham số trong Mask Interaction

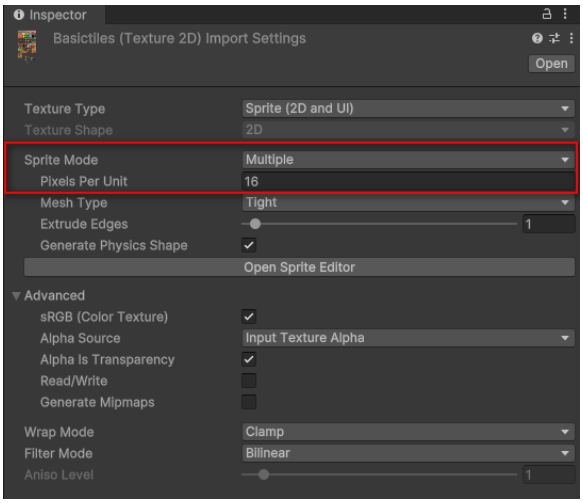
6. Xây dựng bản đồ bằng Tilemaps và Tile Palettes

Tilemap là một công cụ được sử dụng để tạo và quản lý bản đồ trong môi trường game 2D. Một Tilemap là một lưới được chia thành các ô đại diện cho một phần của môi trường game, chẳng hạn như đất đai, tường, nước, hoặc các đối tượng khác. Tilemap thường kết hợp với Tile Assets chính là Texture hoặc Sprite. Tilemap còn cung cấp các tính năng như xử lý va chạm, xác định vị trí của các ô, và tương tác với chúng trong quá trình chạy Game.

Để bắt đầu sử dụng Tilemap, chúng ta cần có Texture hoặc Sprite, có thể tự tạo Texture bằng các công cụ xử lý ảnh hoặc tải các Sprite sẵn có tại trang https://opengameart.org/ (Hình 6.25)

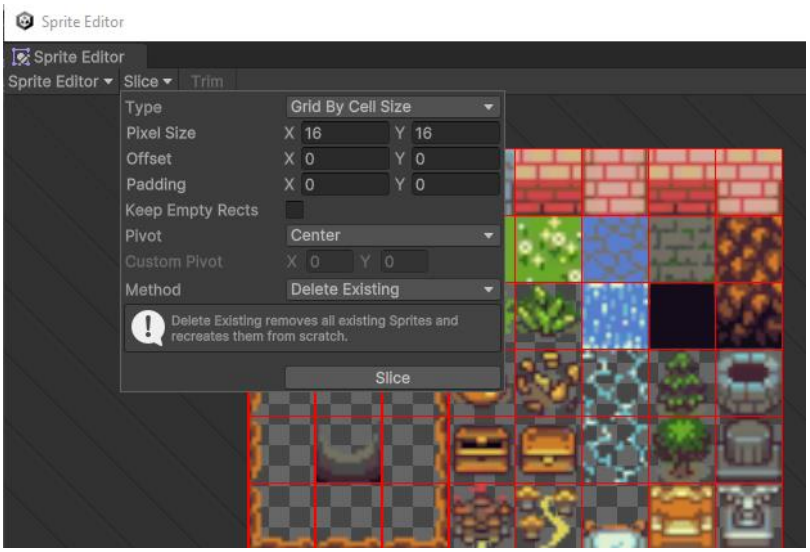
Hình 6.25. Giao diện trang web OpenGameArt

Sau khi tải về, đưa Texture vào trong Project, thiết lập thuộc tính Sprite Mode là *Multiple*, Pixels Per Unit là 16, chọn Apply (Hình 6.26). Các Texture áp dụng cho Tilemap thường thiết lập các ô có độ lớn là 16x16, nên ta cần thiết lập giá trị này để khi cắt sẽ cắt được các đối tượng một cách hợp lý.



Hình 6.26. Thiết lập thuộc tính cho Texture Tilemap

Nhấn Open Sprite Editor để mở Sprite Editor, vào Slice, trong phần Type, chọn ***Grid By Cell Size***, chọn tiếp Pixel Size là **16x16** (Hình 6.27), nhấn nút Slice để tiến hành cắt Texture thành các Sprite theo kích thước đã chọn.

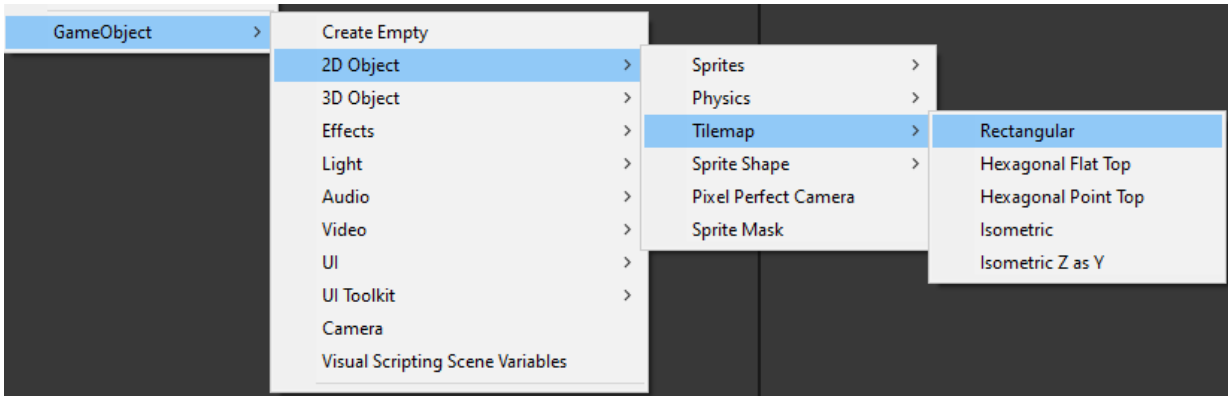


Hình 6.27. Cắt Texture thành các Sprite theo kích thước 16x16

Sau khi đã hoàn thành việc cắt, nhấn nút Apply để chuyển ảnh thành các Sprite, sau đó tắt Sprite Editor. Kết quả là trong Project, từ một Texture ban đầu, Unity đã cắt thành các Sprite theo các đối tượng có kích thước 16x16 như đã thiết lập (Hình 6.28).

Hình 6.28. Các Sprite sau khi cắt

Sau khi cắt thành các Sprite, bước tiếp theo là tạo bản đồ cho Game. Unity hỗ trợ tạo bản đồ bằng công cụ Tilemap, được tạo ra như trong Hình 6.29.

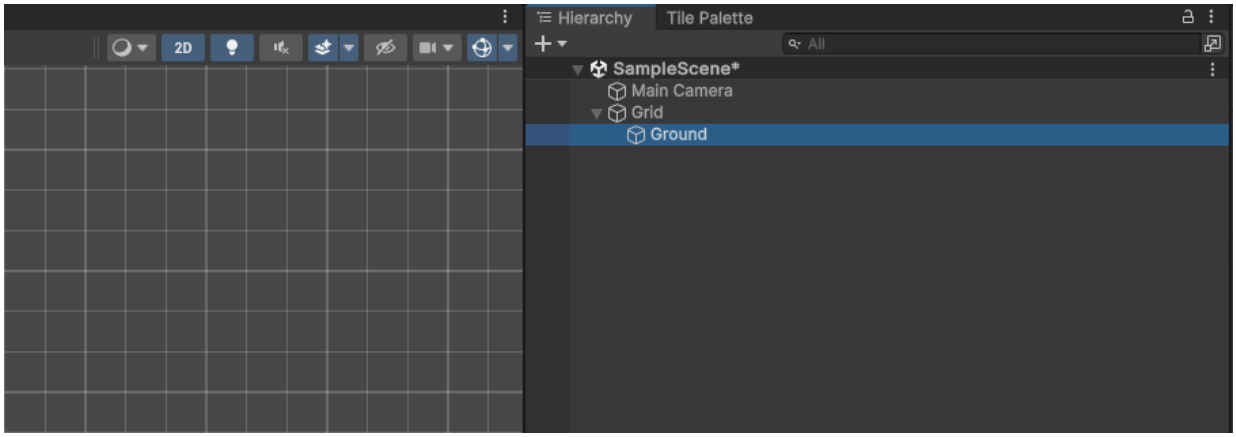


Hình 6.29. Cách tạo Tilemap trong Unity

Tilemap có 4 loại:

- ***Rectangular***: Tạo bản đồ bằng lưới hình chữ nhật, đây là loại thường dùng vì nó tương ứng với kích thước của các Sprite.
- ***Hexagonal Flat Top***: Tạo bản đồ bằng lưới dạng hình lục giác phẳng
- ***Hexagonal Point Top***: Tạo bản đồ bằng lưới lục giác dạng điểm
- ***Isometric***:
- ***Isometric Z as Y***:

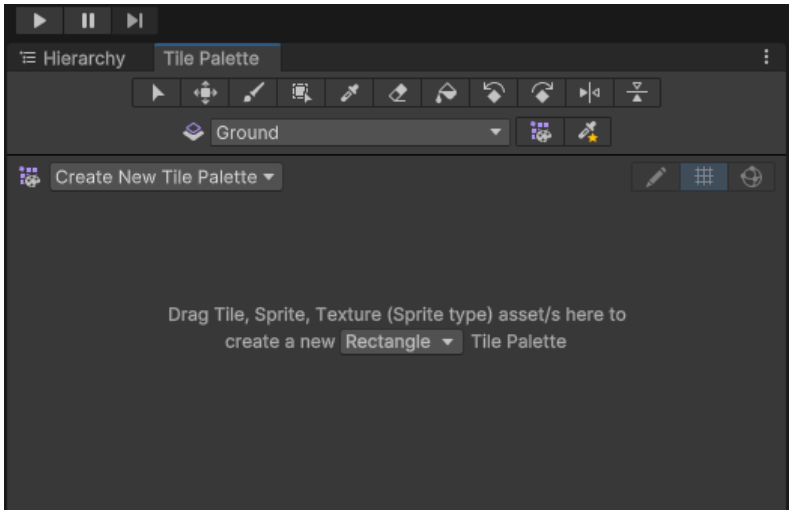
Trong ví dụ này, ta tạo một Tilemap bằng cách chọn Rectangular, Unity sẽ thêm vào một đối tượng lưới Grid và một Tilemap đếu tiên là Tilemap với các thành phần mặc định, đổi tên thành Ground như trong Hình 6.30. Lưu ý, có thể tạo nhiều Tilemap như Groud, Wall, Trees… để có thể làm cho bản đồ sinh động hơn.



Hình 6.30. Tạo một Tilemap tên là Ground.

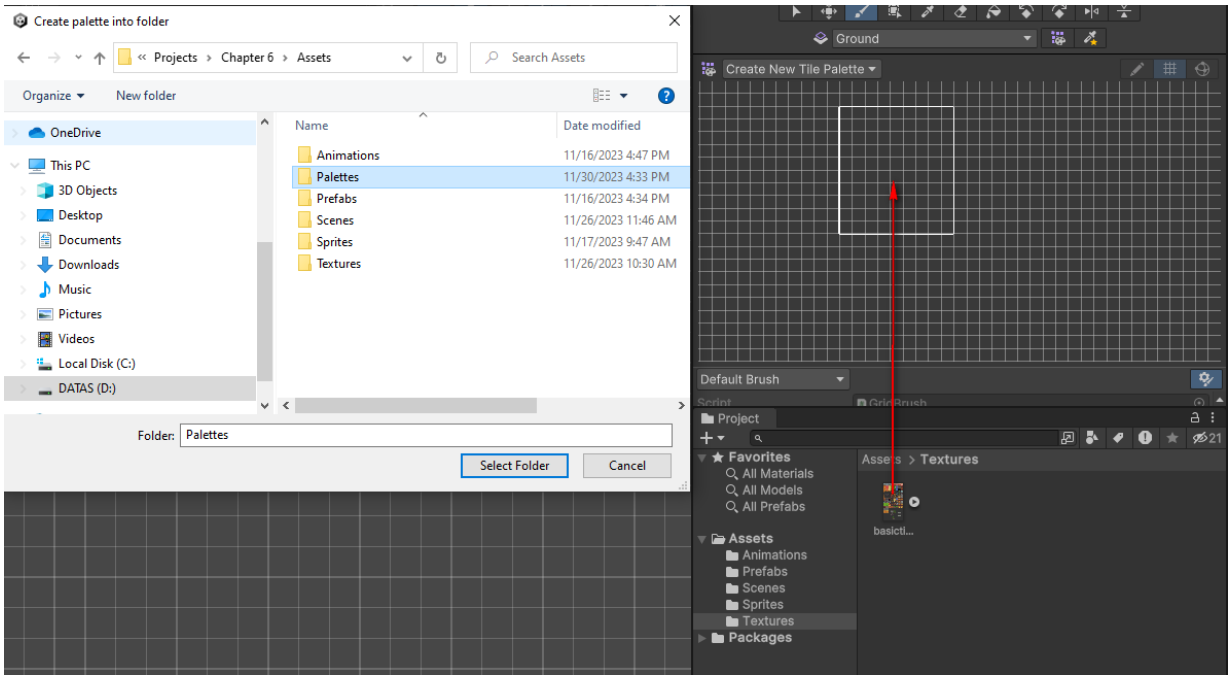
Để hỗ trợ tạo bản đồ một cách nhanh chóng, Unity hỗ trợ công cụ là Tile Palette, công cụ này có các chức năng giúp tạo bản đồ nhanh chóng và hiệu quả.

Để dùng công cụ, vào menu Window, chọn 2D, chọn Tile Palette, đưa cửa sổ Palette đến vị trí phù hợp (Hình 6.31). Khi tạo mới, Tile Palette đã mặc định lựa chọn Tilemap là Ground làm bản đồ mặc định, nếu muốn tạo bản đồ khác, có thể lựa chọn chức năng Create New Tile Palette.



Hình 6.31. Công cụ Tile Palette

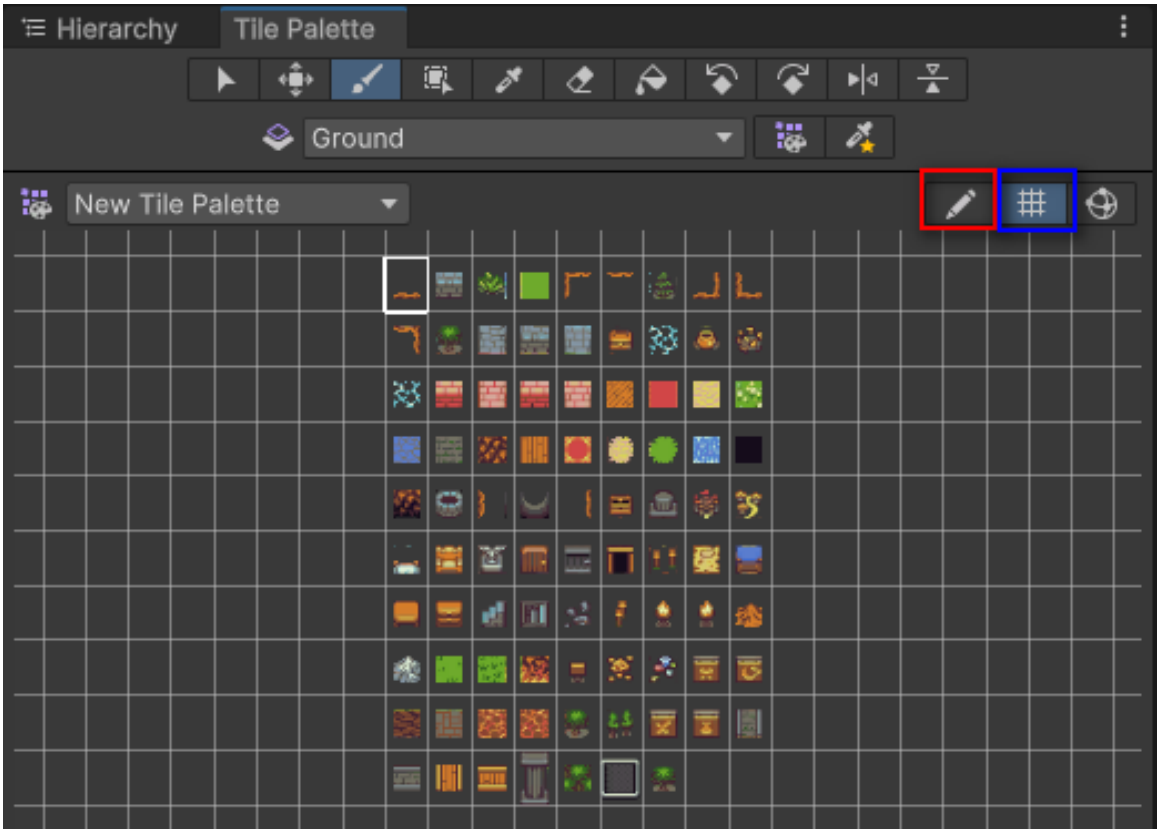
Tiếp theo, tìm đến thư mục chứa các Sprite đã tạo ở bước trên, nhấn chuột và thả vào vùng Tile Palette, hệ thống sẽ yêu cầu lựa chọn một thư mục để chứa các Palette (cần tạo mới thư mục này), chọn thư mục rồi nhấn Select Folder, hệ thống sẽ nhập toàn bộ Sprite vào trong thư mục đó (Hình 6.32), đồng thời mỗi ô trong lưới chứa một Sprite (Hình 6.33).



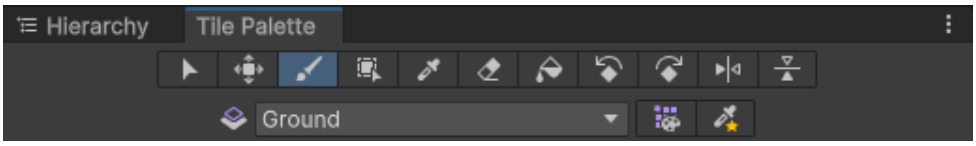
Hình 6.32. Đưa Sprite vào trong Palette

Tile Palette có hai chế độ là Edit và Visibility, có thể bật tắt các chế độ này:

- **Edit**: Dùng để thay đổi các đối tượng trên Palette như xóa, di chuyển…
- **Visibility**: Dùng để đưa các Sprite ra màn hình giao diện, cho phép thực hiện các chức năng bằng bộ công cụ của Palette.



Hình 6.33. Các Sprite đã được đưa vào Palette và 2 chế độ là Edit và Visibility



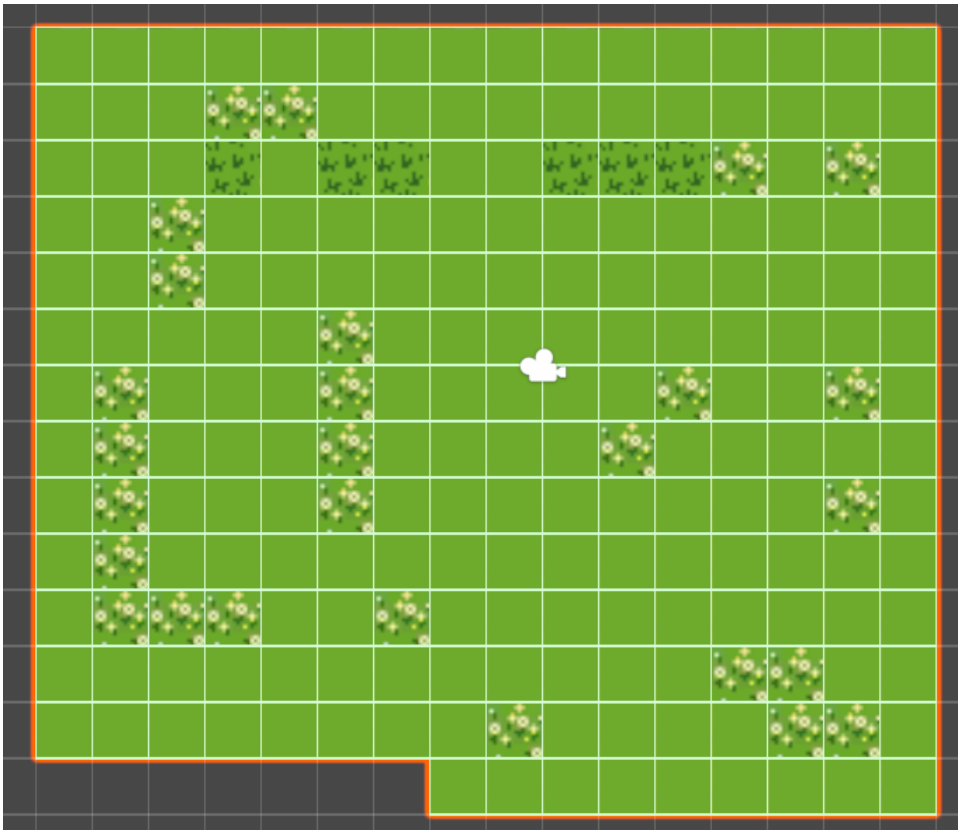
Hình 6.34. Bộ công cụ của Palette

Bộ công cụ của Palette bao gồm các chức năng như sau:

- ***Select an area of the grid*** (): Chức năng này cho phép lựa chọn một ô hoặc một vùng để thực hiện một thao tác khác như xóa hay vẽ ra màn hình
- ***Move selection with active brush*** (): Chức năng này cho phép di chuyển lựa chọn với một cây cọ đang kích hoạt sẵn.
- ***Paint with active brush*** (): Chức năng này cho phép vẽ các Sprite ra màn hình bằng cách lựa chọn Sprite rồi vẽ ra màn hình giao diện.
- ***Paint a filled box with active brush ()***: Hai chức năng này cho phép chọn lựa một vùng bao gồm nhiều ô và vẽ vùng đó ra màn hình.
- ***Erase with active brush ()***: Chức năng này cho phép xóa đối tượng đã vẽ
- ***Flood fill with active brush ()*:** Chức năng này cho phép tô màu cho một vùng của đối tượng khi có biên
- ***Rotates the content of the brush ()***: Hai chức năng này cho phép xoay đối tượng theo chiều kim đồng hồ hoặc ngược chiều kim đồng hồ
- ***Flip the content of the brush ()***: Hai chức năng này cho phép lật đối tượng từ trái qua phải hoặc từ trên xuống dưới.

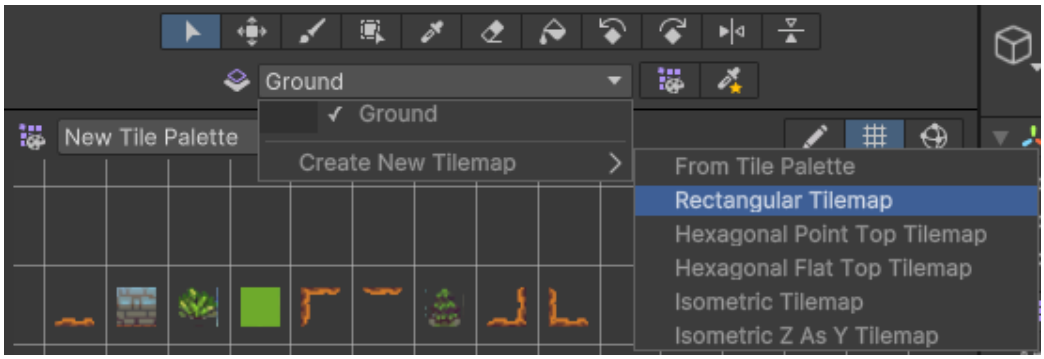
Bằng cách sử dụng các công cụ phù hợp, người dùng có thể vẽ các đối tượng ra màn hình để tạo thành bản đồ cho game (Hình 6.35). Lưu ý là trên Tile Palete có hai chế độ là Edit và

Visibility, nếu bật chế độ Edit, các công cụ này có thể dùng để chỉnh sửa trên chính Tile Palette với các tính năng tương ứng.



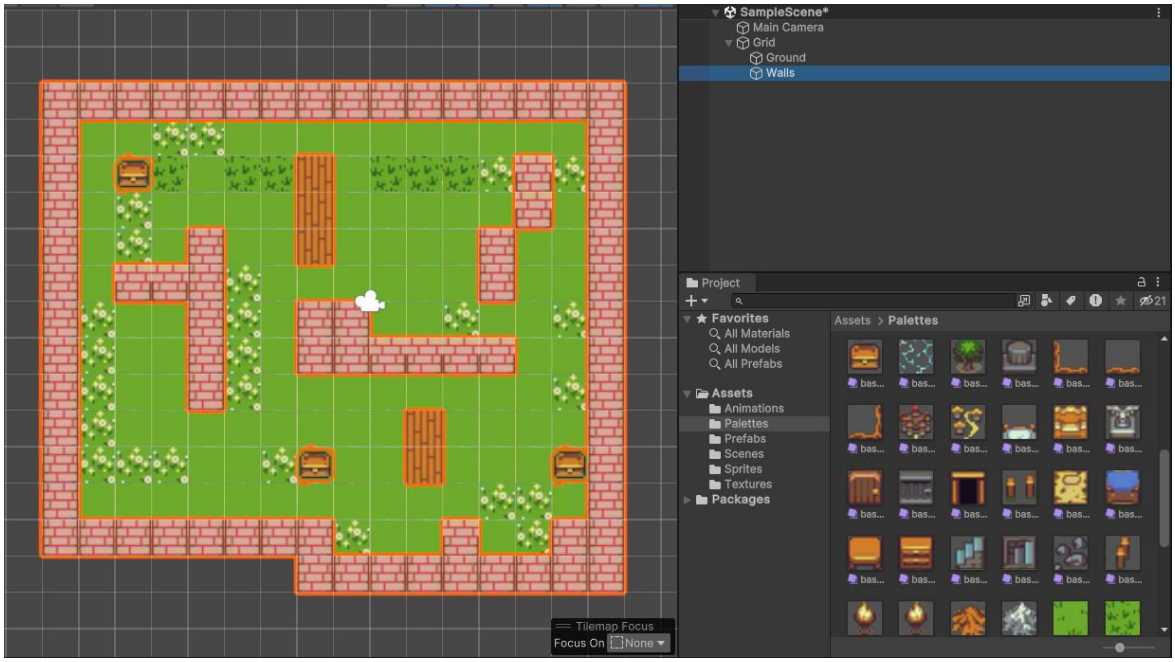
Hình 6.36. Bản đồ được tạo ra bằng công cụ Tile Palette

Ngoài ra, trong một map nếu muốn tạo thêm các Tilemap thì có thể thực hiện bằng cách bấm chọn vào ô chứa các Titlemap, chọn Create New Tilemap (Hình 6.37) và thực hiện như các bước ở trên. Ví dụ, ngay trong chính một map, có thể tạo thêm các Tilemap như Ground, Trees, Animal… để giúp cho bản đồ thêm phần sinh động và có chiều sâu.



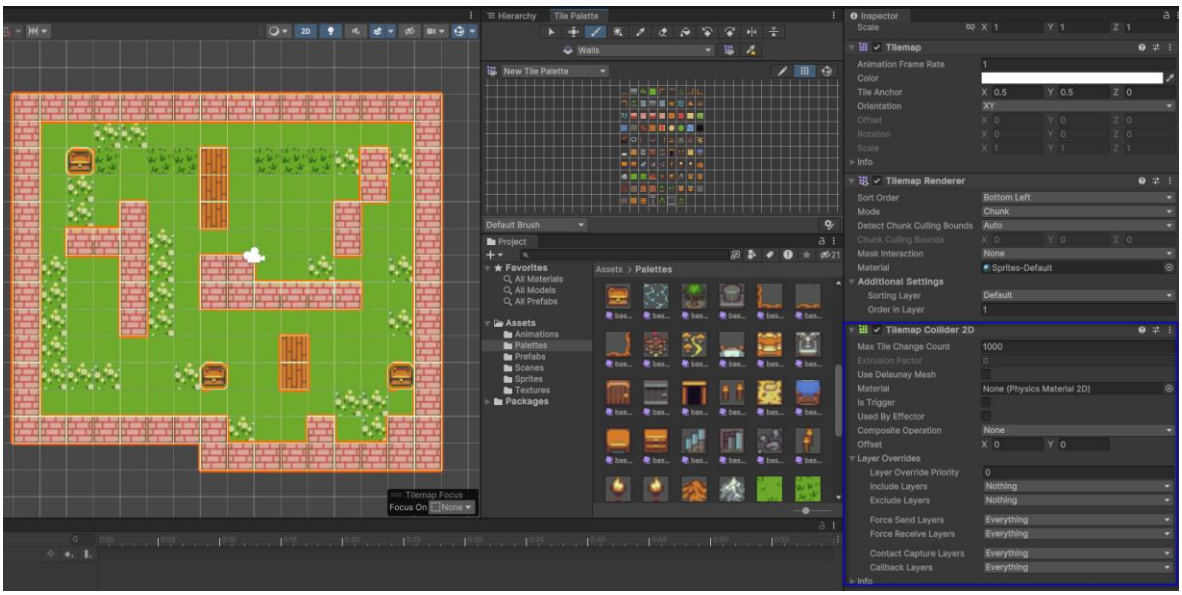
Hình 6.37. Tạo mới Tilemap

Khi tạo mới một Tilemap, cần lưu ý trong phần Inspector, tại thành phần Tilemap Renderer, có thuộc tính Order in Layer, thiết lập giá trị cho thuộc tính này để đặt chiều sâu hoặc thứ tự ưu tiên hiển thị cho các Tilemap trên màn hình game. Hình 6.37 tạo ra hai Tilemap là Ground và Walls, Ground chỉ chứa nền và hoa cỏ (giá trị Order in Layer là 0), trong khi đó Walls (giá trị Order in Layer là 1) chính là các bức tường. Mục đích của việc tạo ra các Tilemap để thiết lập va chạm của nhân vật cho các đối tượng, sẽ được đề cập ở phần tiếp theo.



Hình 6.37. Hai Tilemap trong một màn hình game

Để ngăn ngừa nhân vật và không cho phép đi xuyên qua tường, cần gắn Collider cho từng viên gạch bằng cách thêm thành phần Tilemap Collider 2D cho Walls (Hình 6.38). Có thể thiết lập một số tham số như Material hoặc Is Trigger cho các bức tường này.



Hình 6.38. Thêm Tilemap Collider 2D cho từng viên gạch

Nếu người dùng muốn tạo mới một Tile Palette, có thể click vào phần New Tile Palette, chọn Create New Tile Palette để tạo mới, Unity yêu cầu nhập tên và cho phép chọn các loại lưới Tile Palette như Rectangle hoặc Hexagon.

7. Kết chương

Trong chương này, chúng ta đã tìm hiểu cách xây dựng nhân vật và xây dựng bản đồ cho nhân vật bằng công cụ Sprite Editor và Tile Paletes. Các công cụ này không những giúp game được sinh động mà còn giúp cho việc tạo ra các bản đồ trong game để từ đó có thể triển khai dễ dàng các hoạt cảnh cho game 2D.

Bài tập

1. Sprite là gì? Có mấy loại Sprite? Trong Unity có thể tạo Sprite bằng cách nào?
2. Thực hành tạo một Multiple Sprite tương tự như Hình 6.3.
3. Sprite khác gì so với Texture?
4. Hãy nêu các chức năng chính của công cụ Sprite Editor?
5. Thực hành cắt ảnh và tạo các Sprite động với Sprite Editor.
6. Thực hành tạo các Animation như Idle, Fly, Run… với một Sprite tải về từ Internet.
7. Sprite Creator là gì? Hãy nêu các loại Sprite được tạo ra từ công cụ này.
8. Công cụ Sprite Mask dùng để làm gì? Hãy thực hành với công cụ này trên Unity.
9. Tilemaps là gì? Tile map có mấy loại?
10. Hãy thực hành tạo bản đồ theo một tài nguyên được tải về từ trang: https://opengameart.org/
11. Palette là gì? Hãy nêu và thực hành các chế độ trong Tile Palette

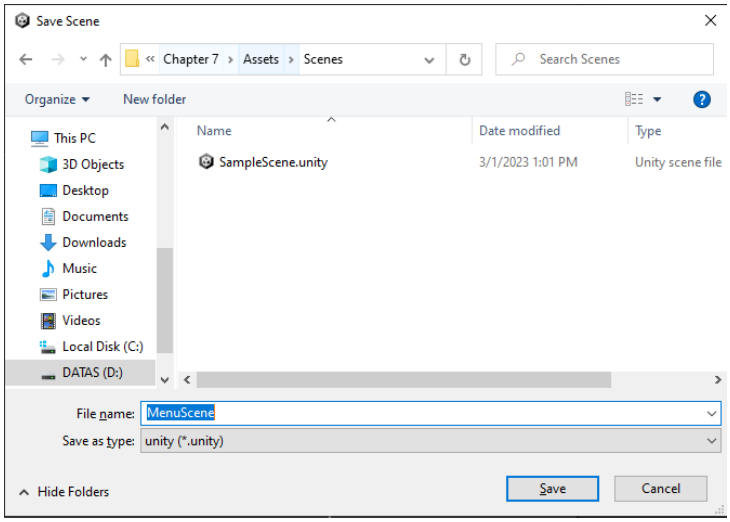
Chương 7. Quản lý màn và thực đơn

Một nội dung không thể thiếu được trong việc tạo ra game đó chính là tạo menu cho người dùng chọn. Ngoài ra trong quá trình hoạt động, game cũng cần phải hiển thị được điểm số hoặc thanh sức khỏe hoặc hiển thị hộp thoại tùy chỉnh, hộp thoại khi thắng… Tất cả những điều này được Unity hỗ trợ bằng khái niệm UI (User Interface) hay còn gọi là giao diện người dùng. Chương này sẽ hướng dẫn người học tạo ra các loại UI được sử dụng trong game.

1. Hệ thống UI trong Unity

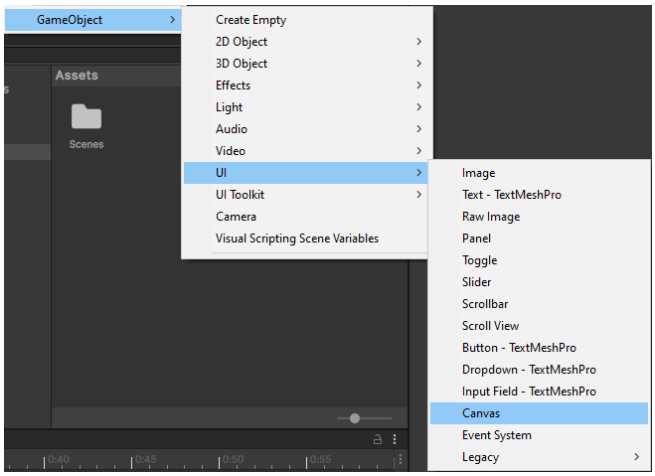
Trong Unity, hệ thống UI được coi như một đối tượng game, tuy nhiên UI thường sẽ được quan sát bằng một Camera ẩn và trên một vùng không gian khác. Khi xử lý trên UI thì giao diện chính sẽ bị thu nhỏ ở phía dưới, tuy nhiên khi hiển thị thì Unity sẽ lồng ghép hai khung cảnh lại với nhau.

Trong tình huống này, ta giả định Game bắt đầu bằng một Menu với ba chức năng như New Game, Options, About & Help. Để tạo được các chức năng này, ta cần tạo một Scene mới (bằng cách vào File, chọn New Scene, đặt tên là MenuScene như Hình 7.1) và lưu vào thư mục Scenes.



Hình 7.1. Tạo mới một màn đặt tên là NewScene

Trong phần Hierachy, thêm Menu bằng cách click phải, chọn Game Object, chọn UI và chọn Canvas (Hình 7.2).



Hình 7.2. Tạo mới UI

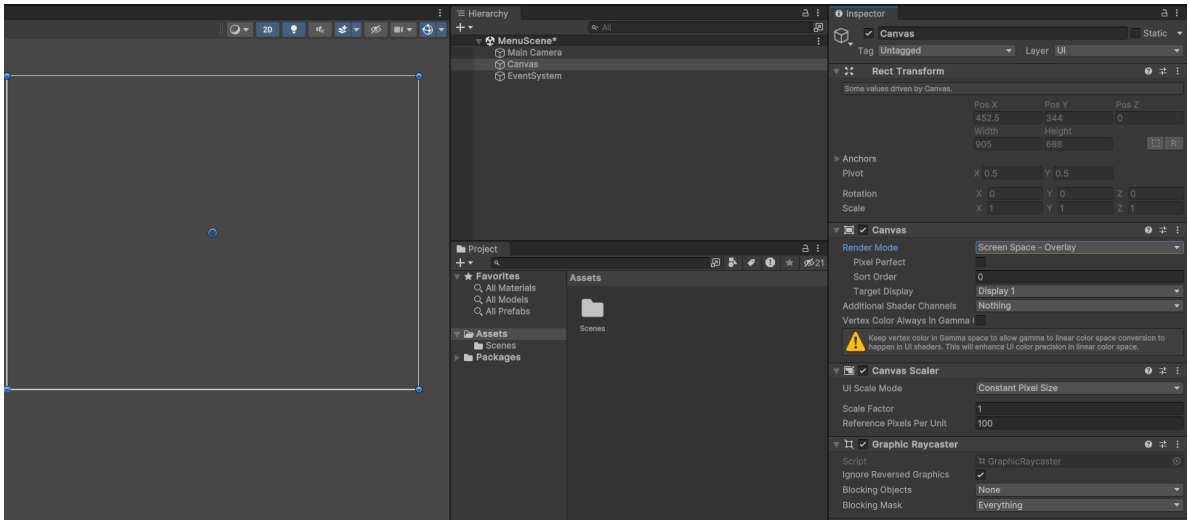
Unity cung cấp các UI chính được mô tả như sau:

- ***Canvas***: Là một đối tượng phẳng dùng để chứa các đối tượng khác, đối tượng này thường được mặc định tạo ra khi ta thêm vào các đối tượng như ảnh, nút bấm…
- ***Panel***: Là mặt phẳng hình chữ nhật, được dùng để chứa các đối tượng khác
- ***Image***: Dùng để chứa một hình ảnh
- ***Raw Image***: Dùng để chứa một hình ảnh bảo toàn được các tính chất của ảnh gốc
- ***Text – TextMeshPro***: Dùng để tạo ra một dòng hoặc một đoạn văn bản
- ***Input Field – TextMeshPro***: Là dạng ô nhập, cho phép người dùng nhập giá trị
- ***Button – TextMeshPro***: Là một nút bấm, trong nút bấm sẽ chứa một văn bản là nhãn của nút bấm
- ***Dropdown-TestMeshPro***: Là dạng ô chọn cho phép người dùng lựa chọn một giá trị trong nhiều giá trị được sổ xuống.
- ***Toggle***: Là dạng nút bấm chuyển đổi trạng thái On-Off
- ***Slider***: Là dạng thanh trượt ngang, dùng để kéo chọn giá trị nằm ngang
- ***Scrollbar***: là dạng thanh trượt dọc, dùng để kéo chọn giá trị lên xuống
- ***Scroll Vew***: Là dạng thanh trượt dùng để đảm bảo khi màn hình quá rộng, sẽ xuất hiện để người dùng kéo.

Ngoài ra còn một số chức năng khác như Even System dùng để tạo ra các sự kiện cho hệ thống hoặc Legacy là các đối tượng dạng cổ điển, các đối tượng này dùng trong phiên bản cũ, hiện đã được thay thế bởi Mesh Pro (không hiểu sao vẫn được Unity giữ lại).

2. Canvas và Panel

Khi tạo mới một đối tượng, đối tượng đó luôn nằm trong một thành phần được gọi là Canvas, đây là một thành phần được chiếu bằng một camera ẩn và khi chạy game thì thành phần này sẽ được lồng ghép để song song với màn hình chính. Click phải, chọn Game Object, chọn UI, chọn Canvas như Hình 7.3. Lúc này có thể dễ thấy là giao diện chính sẽ bị thu nhỏ về góc dưới-trái của màn hình.

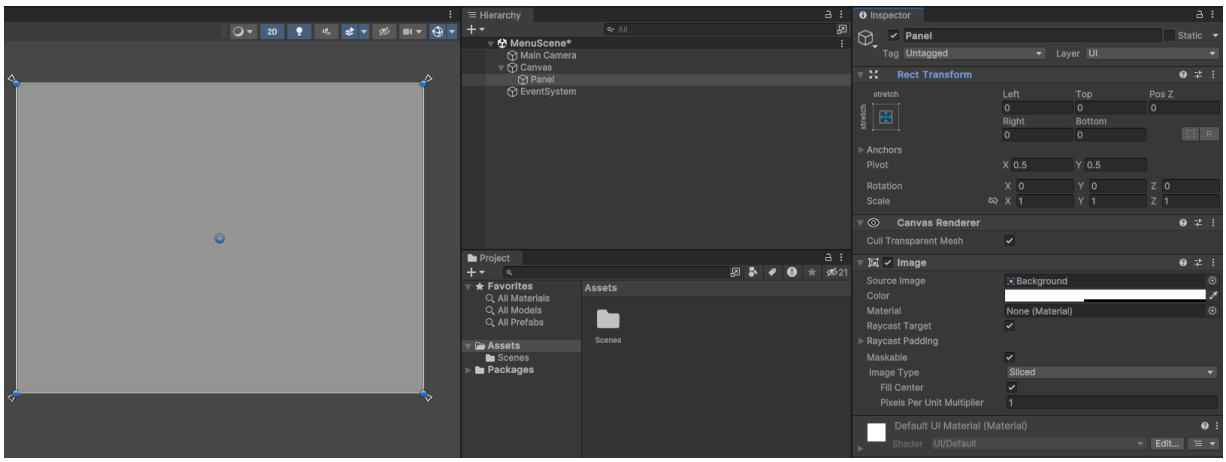


Hình 7.3. Tạo một Canvas

Canvas không cho thay đổi kích thước, mặc định luôn là kích thước của màn hình Game. Canvas có một số thuộc tính cơ bản như sau:

- ***Render Mode***: Là chế độ hiển thị, ở đây mặc định là chế độ Screen Space – Overlay là chế độ pha trộn với màn hình chính như giải thích ở trên.
- ***Sort Order***: Là thứ tự hiển thị, có thể có nhiều Canvas và thuộc tính này sẽ sắp theo thứ tự.
- ***Target Display***: Là màn hình hiển thị, có thể chọn màn hình khác để hiển thị (dùng trong trường hợp Game có nhiều màn hình).

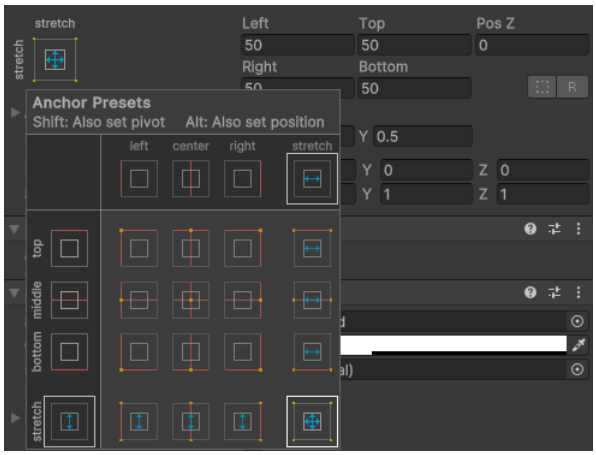
Tiếp tục click chuột phải lên Canvas, chọn UI, chọn Panel, lúc này một Panel sẽ được tạo ra trong Canvas, nhìn vào Inspector, có thể thấy có hai thành phần chính được thêm vào là Rect Transform và Image. Thành phần Rect Transform cho phép thay đổi kích thước, tọa độ hiển thị của Panel, còn thành phần Image thì cho phép thêm và thiết lập ảnh vào trong Panel này (Hình 7.4).



Hình 7.4. Thành phần Panel

Một số thuộc tính chính trên hai thành phần như sau

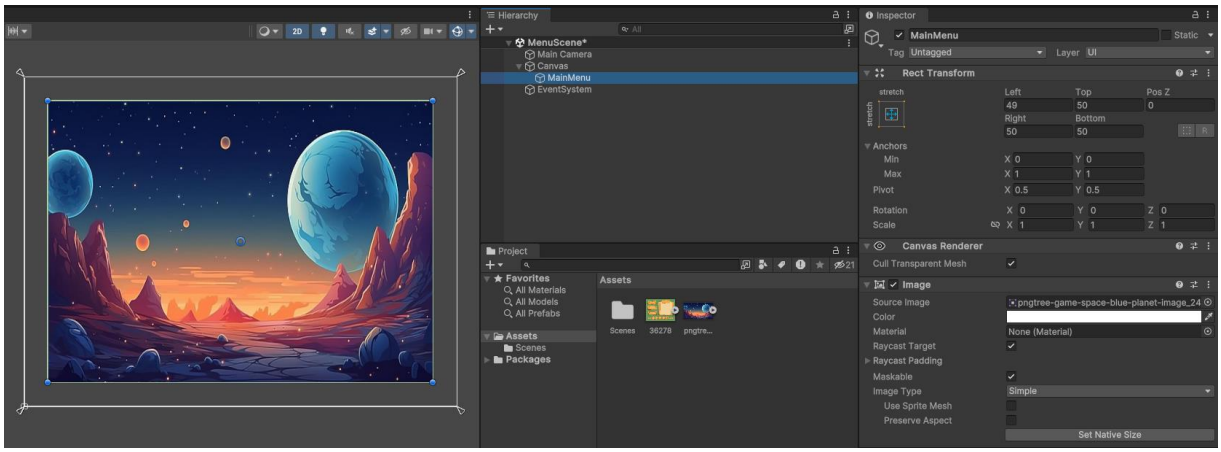
- **Stretch**: Là thuộc tính quy định khoảng cách giữa phần tử chứa với Panel. Khoảng cách được quy định bởi các giá trị trong Left, Top, Right, Bottom. Lưu ý là các khoảng cách này được thiết lập dựa trên Pivot của đối tượng, có thể thay đổi chế độ Pivot bằng cách bấm chuột lên thuộc tính Stretch và nhấn phím Alt, khi đó vị trí của Panel có thể thay đổi và các thuộc tính Left, Top, Right, Bottom sẽ có giá trị tùy vào Pivot (Hình 7.5).



Hình 7.5. Các tham số của Stretch khi bấm chuột vào

- **Anchors**: Là các điểm neo, dùng để giữ đối tượng trong Canvas.
- **Pivot**: Là tọa độ của Panel, mặc định là 0.5, 0.5 tức là tọa độ nằm giữa Panel.
- **Rotation & Scale**: Trục quay đối tượng và co dãn đối tượng, cho phép quay hoặc co dãn kích thước đối tượng.
- **Rotation & Scale**: Trục quay đối tượng và co dãn đối tượng, cho phép quay hoặc co dãn kích thước đối tượng.
- **Source Image**: Thuộc tính này cho phép thiết lập hình nền cho Panel. Hình nền được chọn trong Project.
- **Color**: Thiết lập màu nền cho Panel. Có thể thiết lập màu trong suốt để hình nền được rõ hoàn toàn
- **Material**: Thuộc tính này cho phép chọn chất liệu cho Panel.

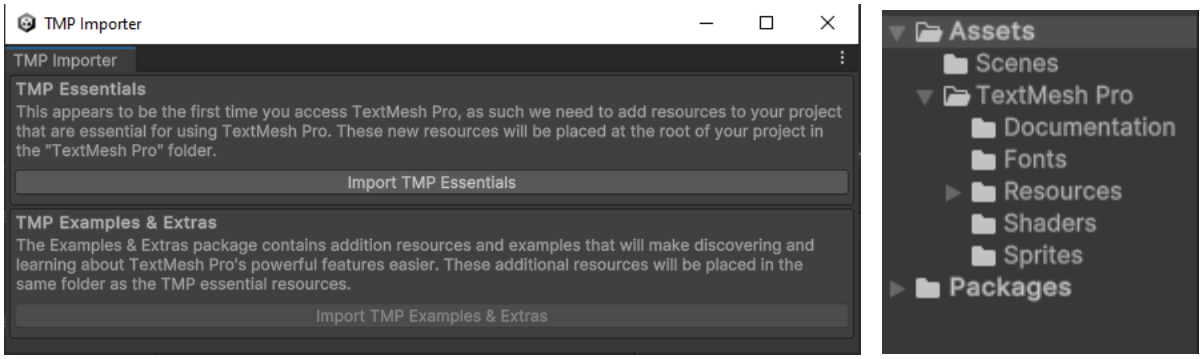
Dựa trên các tham số này, người dùng cho thể tạo ra một hộp thoại phù hợp bằng cách thiết lập màu sắc, hình ảnh kích thước của Panel (Hình 7.6).



Hình 7.6. Một Canvas và Panel đưa vào một ảnh nền và màu trong suốt

3. Thành phần hiển thị văn bản và hình ảnh

Để hiển thị một dòng văn bản trên Panel, Unity hỗ trợ thành phần TextMeshPro – Text (UI), đây là một đối tượng mới, dùng để thay thế đối tượng Text (có trong Legacy). TextMeshPro – Text (UI) có thể được thêm vào Panel bằng cách click phải, chọn UI, chọn Text TextMeshPro – Text (UI), nếu là lần đầu tiên tạo Text Mesh Pro thì một hộp thoại hiện lên để yêu cầu nhập các tài nguyên vào hệ thống, nhấn nút Import TMP Essentials để thêm các tài nguyên vào trong Project (Hình 7.7).

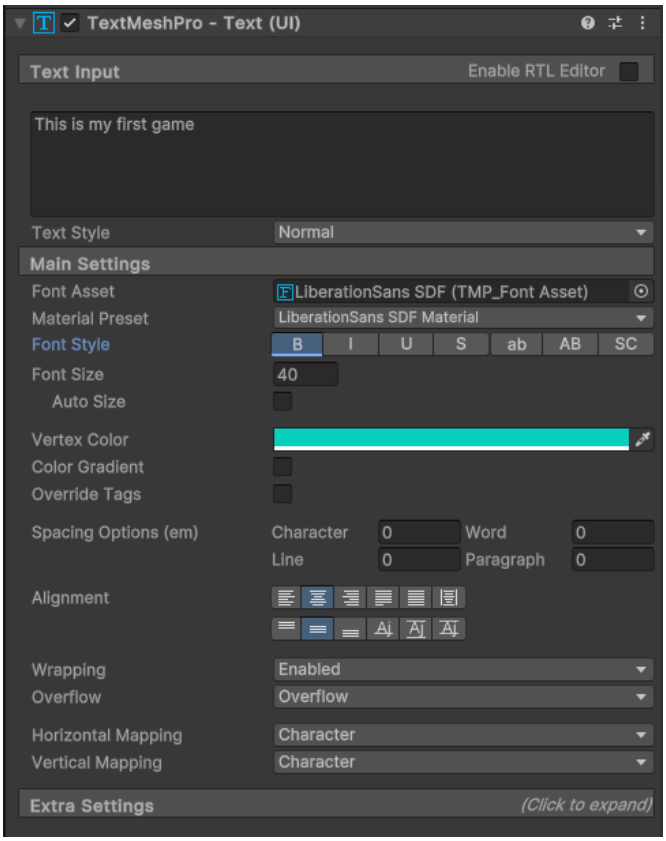


Hình 7.7. Thêm tài nguyên cho lần đầu tạo UI dạng Text Mesh Pro

Khi được thêm vào Panel, đối tượng này ngoài thành phần Rect Transform như mô tả ở trên, thì nó cũng cho phép chỉnh sửa, trang trí nội dung liên quan đến văn bản như:

- **Text input**: Văn bản hiển thị lên Button
- **Font Asset**: Cho phép chọn Font chữ hiển thị, font chữ có thể được đưa vào trong Project.
- **Font Style và Font Size:** Cho phép trang trí văn bản
- **Vertex Color:** Màu chữ
- **Alignment:** Căn chỉnh văn bản**.**
- **….**

Một số thuộc tính khác được biểu diễn như trong Hình 7.8, khi thiết kế Game, tùy vào từng chức năng để chọn các thuộc tính cho phù hợp.

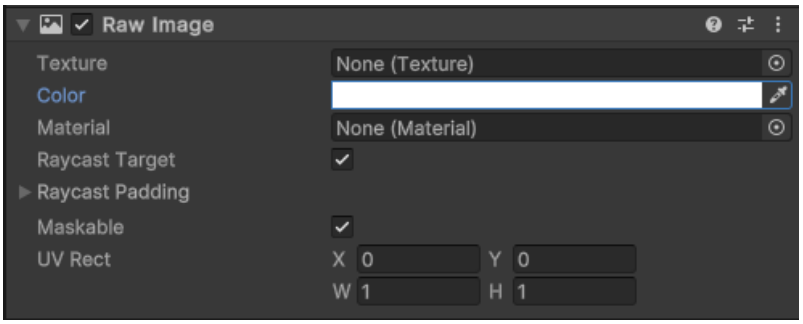


Hình 7.8. Một số thuộc tính của TextMeshPro – Text (UI)

Một đối tượng không thể thiếu trong trang trí UI cho Game, đó chính là Image. Thành phần này cũng được thêm vào bằng cách click chuột phải, chọn UI, chọn Raw Image. Đối tượng này có một thành phần là Raw Image dùng để quản lý Image do người dùng nhập vào. Một số thuộc tính chính của thành phần Raw Image như:

- **Texture**: Là hình ảnh gốc cần hiển thị lên màn hình
- **Color**: Màu sắc cho ảnh, nếu không muốn ảnh bị ảnh hưởng bởi màu sắc, có thể để thuộc tính A (Alpha) về giá trị 0.
- **Material:** Là thành phần cho phép đưa vào chất liệu của ảnh.
- **Raycast Target:** Thành phần này có cho phép là đối tượng của tia va chạm hay không.

Và một số thành phần khác như trên Hình 7.9



Hình 7.9. Các thuộc tính của thành phần Image Raw.

Hình 7.10 cho thấy thành phần TextMeshPro-Text(UI) và thành phần Image Raw được thiết kế khi chạy trên môi trường Unity.

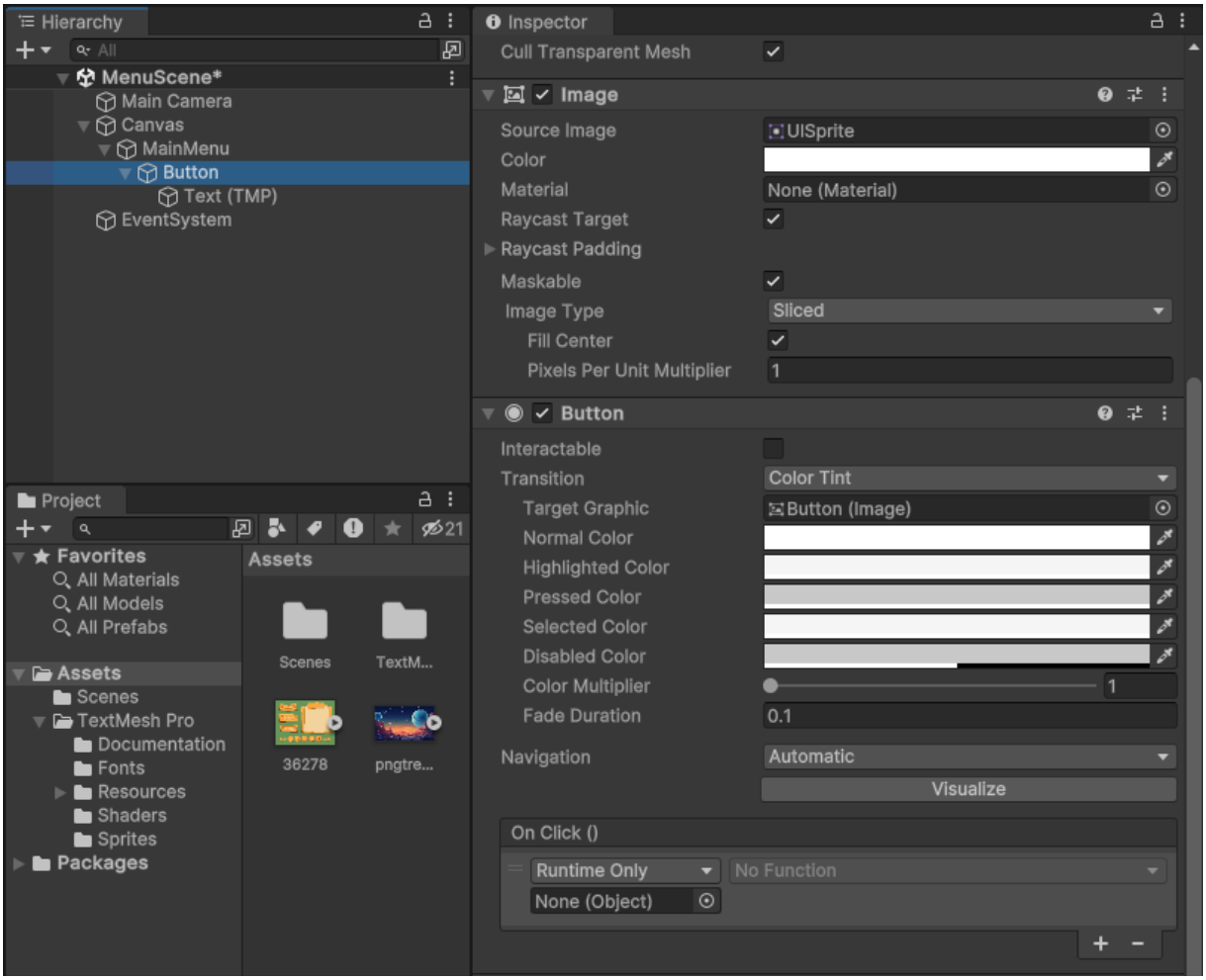
Hình 7.10. Hiển thị văn bản và hình ảnh trong UI của Unity

4. Thành phần Button

Unity hỗ trợ người dùng thiết kế Button như Button (trong Legacy) hoặc *Button – Text Mesh Pro* (loại này mới và cần phải thêm tư viện). Button cho phép người dùng nhấn phím để thực hiện một chức năng nào đó của Game. Để tạo *Button – Text Mesh Pro*, click phải lên Canvas (hoặc Panel đã tạo trước đó), chọn UI, chọn Button – Text Mesh Pro, nếu là lần đầu tiên tạo Text Mesh Pro thì một hộp thoại hiện lên, nhấn nút Import TMP Essentials để thêm các tài nguyên liên quan đến Button vào trong Project (Hình 7.7).

Mỗi khi Button thêm vào, sẽ có hai đối tượng được thêm tự động là Button và Text. Button dùng để biểu diễn các tính chất của một nút còn Text để biểu diễn các tính chất của văn bản trong nút bấm đó. Thành phần chính của Button ngoài hai thành phần cơ bản là Rect Transform và Image (có các tính chất như Panel nêu ở trên), còn có một thành phần nữa là Button với các tính chất như sau:

- ***Interactable***: Cho phép tương tác hay không, nếu không chọn sẽ là một Button bình thường, không cho phép hiển thị các hiệu ứng như rê chuột vào hoặc bấm chuột.
- ***Transition***: Thuộc tính này quy định các loại hiệu ứng như None (không có hiệu ứng), Color Tint (hiệu ứng cho màu sắc), Sprite Swap (hiệu ứng cho các Sprite), Animation (hoạt cảnh). Tùy vào cách chọn loại hiệu ứng mà Unity có thể cung cấp thêm các thuộc tính như: Normal (mặc định), Pressed (khi được nhấn), Highlighted (trang trí), Selected (khi được chọn), Disabled (nút dạng vô hiệu hóa)… (Hình 7.11)



Hình 7.11. Một số thuộc tính của thành phần Button

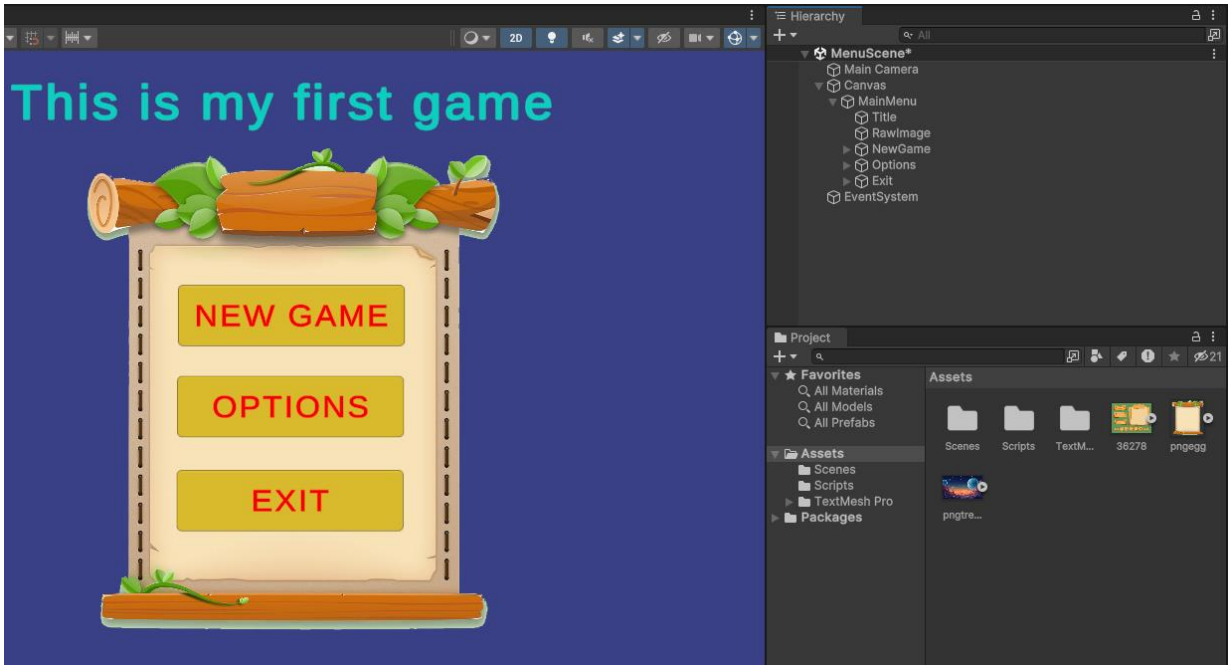
Ngoài ra thành phần Button còn có sự kiện On Click () là sự kiện xảy ra khi người dùng bấm bút, để xử lý trong phần mã nguồn. Khi muốn thêm sự kiện, cần bấm vào dấu +, kéo đối tượng Game vào phần None (Object) và chọn hàm tương ứng trong phần No Function. Lưu ý là đối tượng Game khi kéo vào phần None (Object) bắt buộc phải là đối tượng Game chứa mã nguồn có hàm cần gọi, nếu mã nguồn chỉ có một hàm (ví dụ hàm LoadGame()) thì có thể gắn mã nguồn vào đối tượng Main Camera, mục đích của phần này là chỉ dùng để lấy hàm số nên đối tượng Game không quan trọng.

Lưu ý rằng, thành phần Button cho phép đưa các vào Sprite để làm các hình cho các sự kiện như Normal, Pressed, Selected…, nên ta có thể thiết kế các hình này trước bằng công cụ Sprite Editor, sau đó cắt thành các Sprite tương ứng, rồi chuyển vào Project và áp dụng cho các thành phần của nút.

Các bước sau đây hướng dẫn việc tạo một Menu Game với các nút đơn giản để khi bấm vào, sẽ thực hiện các chức năng tương ứng.

Bước 1: Tạo một Panel đặt tên là MainMenu, thiết lập các tham số như Left, Right, Top, Button và màu phù hợp.

Bước 2: Thêm các Button như New Game, Options, Exit là con của MainMenu và thiết lập các tham số liên quan phù hợp như trong Hình 7.12. Lưu ý, để sinh động hơn, có thể dùng các Sprite để thay thế các màu cho Button.



Hình 7.12. Thiết kế Game Menu

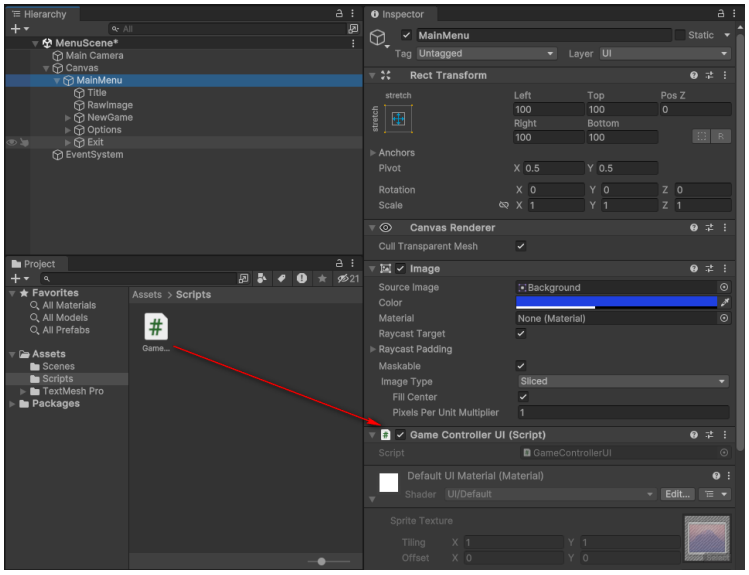
Bước 3: Tạo một Script có tên là GameControllerUI.cs, đưa vào trong Project (nên tạo một thư mục có tên là Scripts và đưa vào đó). Xóa hai hàm là Update và Start, viết 3 hàm là NewGame(), Option() và ExitGame() với các nội dung như trong Hình 7.13.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.SceneManagement;
        Unity Script (2 asset references) | 0 references
5   public class GameControllerUI : MonoBehaviour
6   {
7       // Hàm này sẽ giúp chạy để mở hộp thoại tùy chọn cho Game
            0 references
8       public void Option()
9       {
10          // Code hàm này sẽ viết sau
11      }
12      // Hàm này sẽ giúp chạy để mở một màn mới
            0 references
13      public void NewGame()
14      {
15          SceneManager.LoadScene("SampleScene");
16      }
17      // Hàm này sẽ thoát game
            0 references
18      public void ExitGame()
19      {
20          Application.Quit();
21          Debug.Log("Thoát game");
22      }
23  }
```

Hình 7.13. Mã nguồn cho New Game và Exit Game

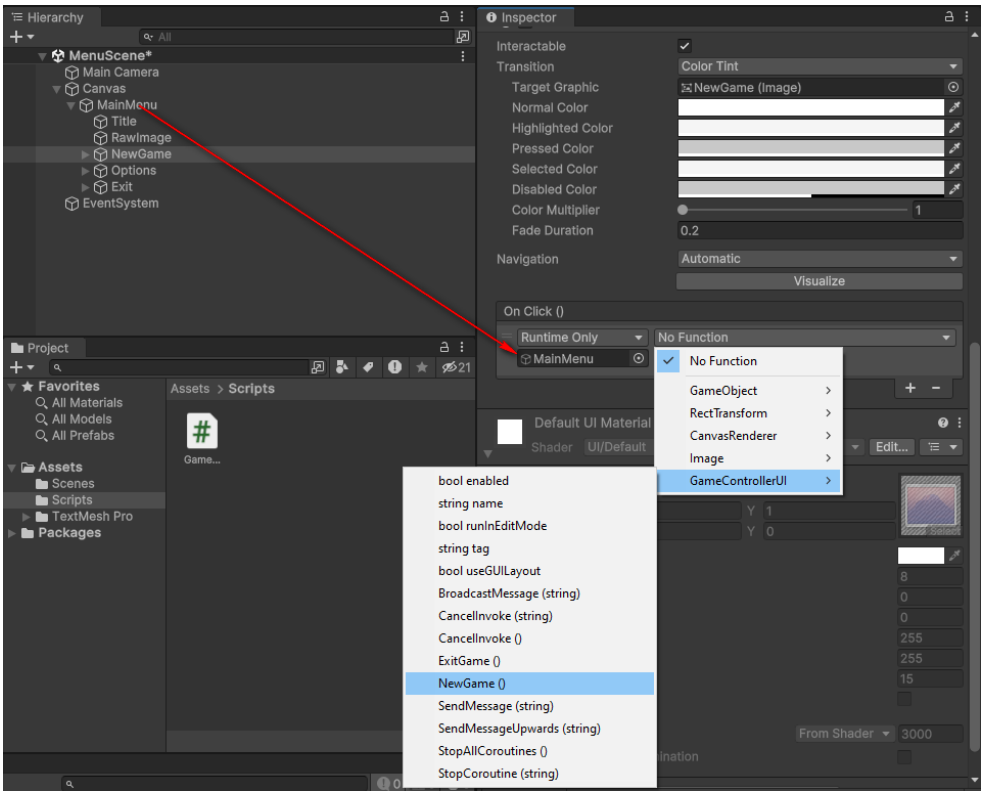
Lưu ý rằng, SampleScene là Scene đã được thiết kế và kéo vào trong phần Scenes In Build của chức năng Build Settings trong Menu File\Build Settings (Xem lại Chương 3); Cần phải thêm vào thư viện như `UnityEngine.SceneManagement` để thực hiện các chức năng như tải Scene hoặc lấy các UI cần thiết

Bước 4: Gắn mã nguồn GameControllerUI cho đối tượng MainMenu (Hình 7.14)



Hình 7.14. Gắn mã nguồn GameControllerUI cho đối tượng Panel

Bước 5: Trong nút New Game, trên Inspector, trong thành phần On Click(). Bấm vào dấu + để tạo một sự kiện mới, nhấn chuột vào Panel, kéo Panel vào phần None (Object), trong phần No Function, bấm chuột để chọn đối tượng GameControllerUI và chọn hàm NewGame() (Hình 7.15).



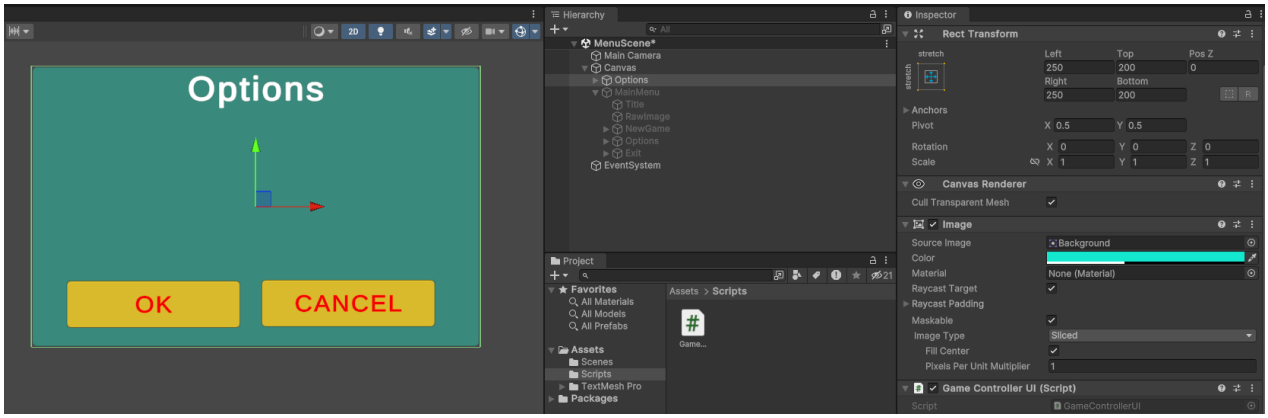
Hình 7.15. Kéo đối tượng cho sự kiện và chọn hàm tương ứng cho nút New Game

Thực hiện tương tự cho nút Exit nhưng lần này là chọn hàm ExitGame(). Chạy game, bấm nút New Game hoặc Exit để xem thành quả. Lưu ý rằng, chức năng ExitGame() chỉ hiển thị chữ "Thoát game" ở màn hình Console mà không dừng chức năng Play, chức năng này sẽ có hiệu quả khi Game đã phát hành.

Bước 6: Thiết kế hộp thoại Options

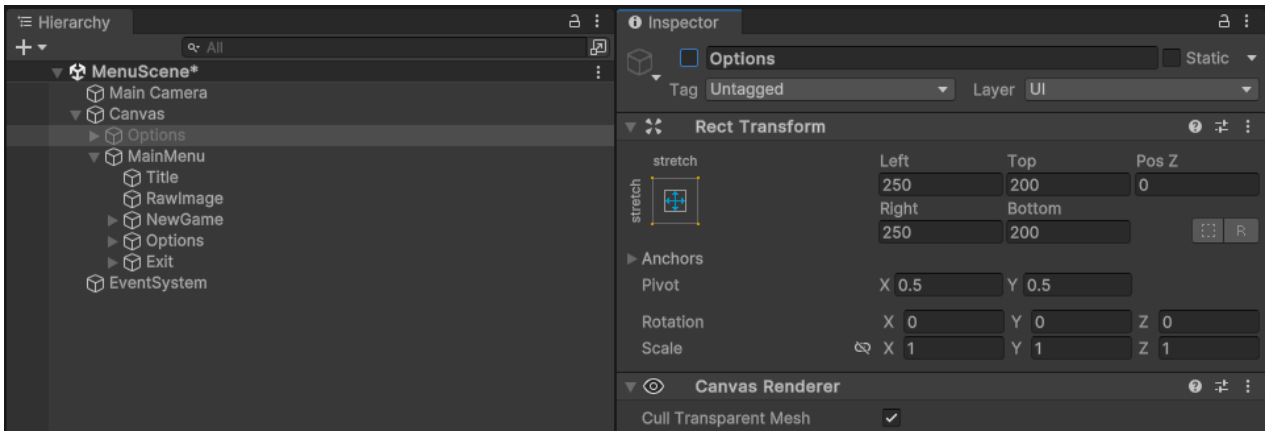
Hộp thoại Options bao gồm các chức năng như tùy chỉnh âm thanh hoặc chọn độ khó, dễ cho game, các chức năng này sẽ được hướng dẫn chi tiết trong các phần sau. Phần này chỉ hướng dẫn cách bật tắt hộp thoại Options.

Tạo một Panel khác đặt tên là Options, thiết kế hai nút là OK và Cancel, thêm vào một TextMeshPro như Hình 7.16. Lưu ý, để nhìn rõ Panel Options mà không bị MainMenu lúc trước che khuất hoặc gây khó chịu, có thể vô hiệu hóa bằng cách tắt dấu kiểm ở trên đầu của Inspector của Panel (Hình 7.16).



Hình 7.16. Tạo hộp thoại Options

Sau khi thiết kế xong, mở lại MainMenu, và tắt hộp kiểm của Options (Hình 7.17), ta sẽ bật lên bằng cách viết mã nguồn ở Bước 7.



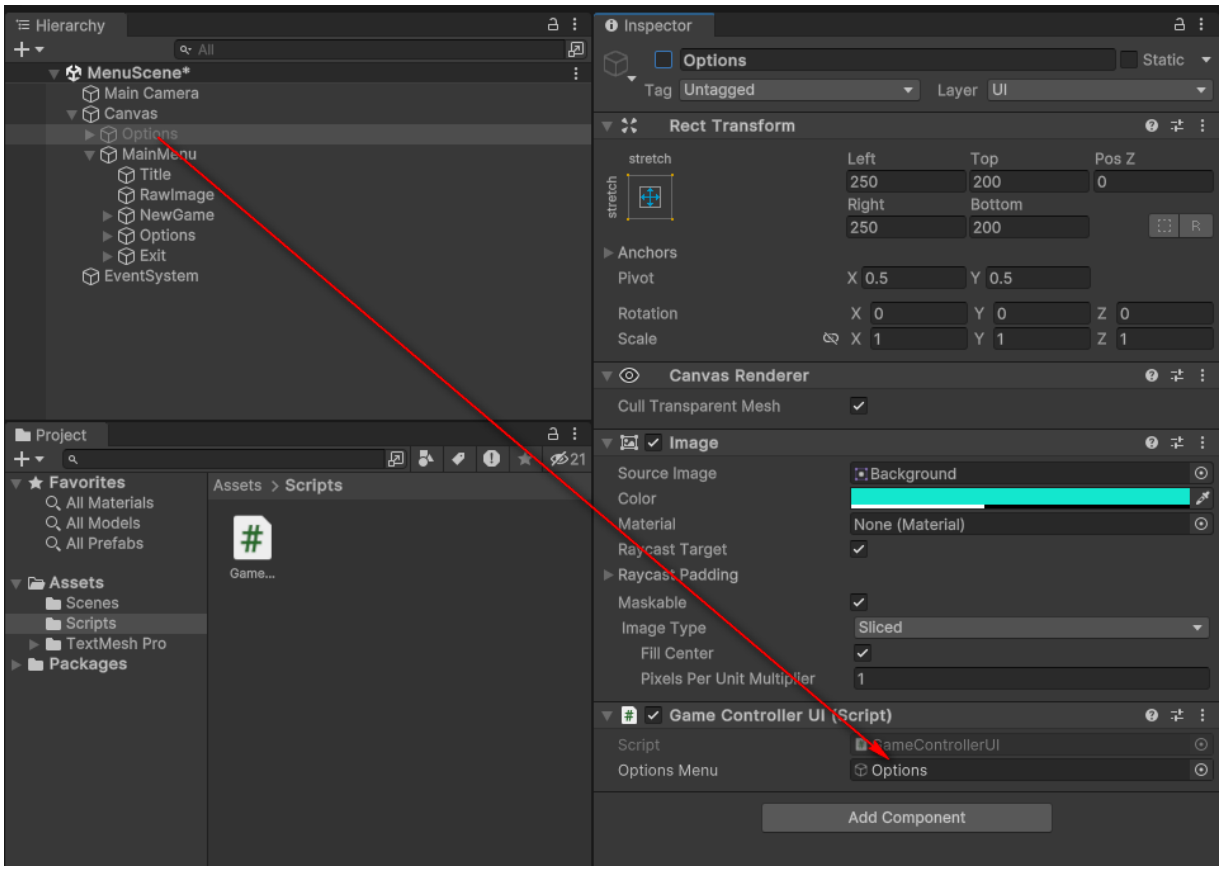
Hình 7.16. Bật loại hộp thoại MainMenu, tắt hộp thoại Options

Bước 7: Trong mã nguồn GameControllerUI, thêm vào đối tượng GameObject và viết các lệnh trong hàm Option() như trong Hình 7.17.

```
Unity Script (1 asset reference) | 0 references
public class GameControllerUI : MonoBehaviour
{
    public GameObject optionsMenu;
    // Hàm này sẽ giúp chạy để mở hộp thoại tùy chọn cho Game
    0 references
    public void Option()
    {
        gameObject.SetActive(false); // Vô hiệu hóa Mainmenu
        optionsMenu.SetActive(true); // Bật hộp thoại Option
    }
```

Hình 7.17. Mã nguồn cho hàm Option()

Bước 8: Chuyển qua Unity, chọn đối tượng MainMenu, kéo đối tượng Options vào trong phần Options Menu của Script (Hình 7.18)



Hình 7.18. Kéo Menu Options vào cho đối tượng Options Menu của Script

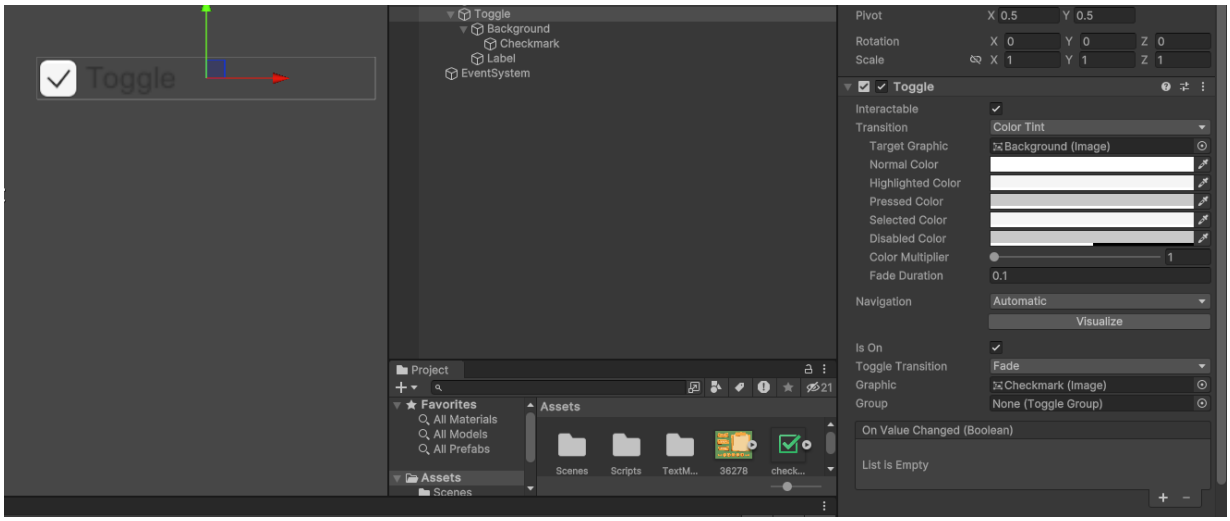
Bước 9: Thêm sự kiện nhấn nút cho Options trong MainMenu như ở **Bước 5**, chạy Game và xem thành quả khi nhấn các nút.

5. Thiết kế hộp thoại Options với Toggle và Slider

Quay lại với hộp thoại Options, ta sẽ dùng các đối tượng như Text, Toggle và Slider để thiết kế hộp thoại cho phép chọn độ khó dễ của Game và tùy chỉnh âm thanh trong Game. Để làm được điều này, cần tìm hiểu qua về đối tượng Toggle và đối tượng Slider.

Đối tượng Toggle là loại nút cho phép chuyển hai trạng thái là bật hoặc tắt, hai trạng thái này tương tự như CheckBox trong lập trình (Hình 7.19). Khi tạo mới một Toggle, Unity sẽ tạo ra 4 đối tượng bao gồm:

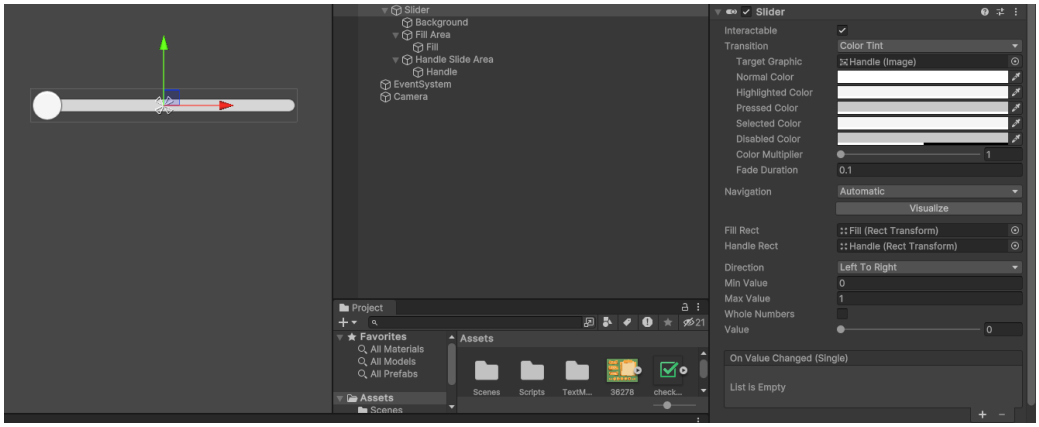
- **Toggle**: Quy định cách trình bày nút chuyển trạng thái, đối tượng này chứa một số thành phần chính như *Normal Color* (màu mặc định), Pressed Color (màu khi nhấn nút), Is On (nút được đánh dấu)…. Toggle cũng quy định về sự kiện On Value Changed để thiết lập sự kiện khi người dùng chọn hoặc bỏ chọn.
- **Background**: Quy định nền của nút, chính là một hình ảnh được đưa vào để làm hình nền cho nút chọn.
- **Checkmark**: Quy định cách thức hiển thị dấu kiểm, cách thức hiển thị dấu kiểm cũng cho phép thay thế bằng một hình ảnh phù hợp, có các thuộc tính tương tự như thuộc tính Image.
- **Label:** Quy định về văn bản hiển thị trên nút, người dùng có thể chỉnh sửa các thành phần của văn bản này như đậm-nhạt, căn chỉnh, font chữ…



Hình 7.19. Các đối tượng game trong Toggle

Đối tượng Slider cho phép thiết kế một thanh trượt để chọn giá trị từ nhỏ đến lớn, khi tạo mới một Slider, Unity sẽ tự động thêm vào một số đối tượng như sau (Hình 7.20):

- **Slider**: Quy định cách trình bày thanh trượt, đối tượng này chứa một số thành phần chính như *Normal Color* (màu mặc định), Pressed Color (màu khi nhấn nút), Min Value (là giá trị tối thiểu), Max Value (giá trị tối đa)…. Slider cũng có quy định về sự kiện On Value Changed để thiết lập sự kiện khi người dùng thay đổi giá trị của thanh trượt.
- **Background**: Quy định nền của thanh trượt, là một hình ảnh làm hình nền cho thanh trượt.
- **Fill Area** và **Fill**: Quy định cách thức hiển thị phần đã được kéo, phần này bao gồm một đối tượng cha (Fill Area) chỉ bao gồm vị trí, và một đối tượng con (Fill) quy định cách thức hiển thị thanh trượt khi đã chọn, có thể thay thế bằng một hình ảnh phù hợp.
- **Handle Side Area** và **Handle:** Quy định về nút dùng để kéo giá trị, có thể thay thế nút này bằng một hình ảnh phù hợp.

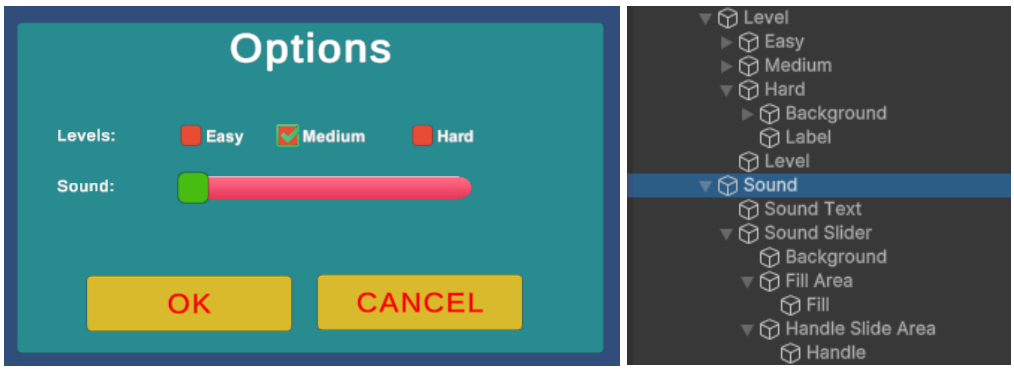


Hình 7.20. Đối tượng Slider và các đối tượng trong đối tượng Slider

Phần sau đây sẽ hướng dẫn cách tạo hộp thoại Options với chức năng chọn độ khó dễ cho Game và thay đổi âm thanh cho Game.

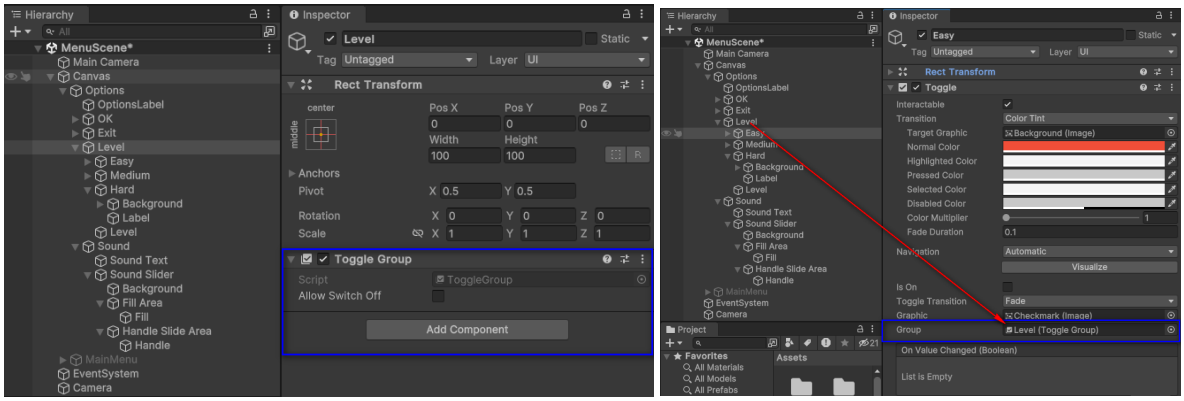
Bước 1: Trong Panel Options, thêm vào một đối tượng Empty, đặt tên là Level với 4 đối tượng con bao gồm 3 Toggle (Easy, Medium, Hard) và một đối tượng Label. Tìm hình ảnh phù hợp, thiết kế như Hình 7.12.

Bước 2: Tiếp tục thêm vào một đối tượng Empty, đặt tên là Sound với hai đối tượng được thêm vào là Sound Text kiểu Label và Sound Slider kiểu Slider. Tìm hình ảnh phù hợp, thiết kế như Hình 7.21.



Hình 7.21. Menu Options với các đối tượng Toggle và Slider

Bước 3: Trong mục Level, để cho phép chỉ chọn một mức độ game trong ba chế độ, click chuột phải lên đối tượng Level, thêm vào một thành phần có tên Toggle Group. Sau đó với mỗi thành phần là Easy, Medium và Hard chọn đối tượng trong thuộc tính Group là Level (Hình 7.22).



Hình 7.22. Thêm thành phần Toggle Group cho Level và thêm vào thuộc tính Group

Chạy Game, sẽ thấy hộp thoại Options chỉ cho phép chọn một trong ba cấp độ là Easy, Medium và Hard.

Bước 4: Tạo một script có tên là OptionMenu.cs, viết một hàm có tên BackToMenu() với các lệnh như sau (lưu ý cần thêm thư viên UnityEngine.UI) (Hình 7.23):

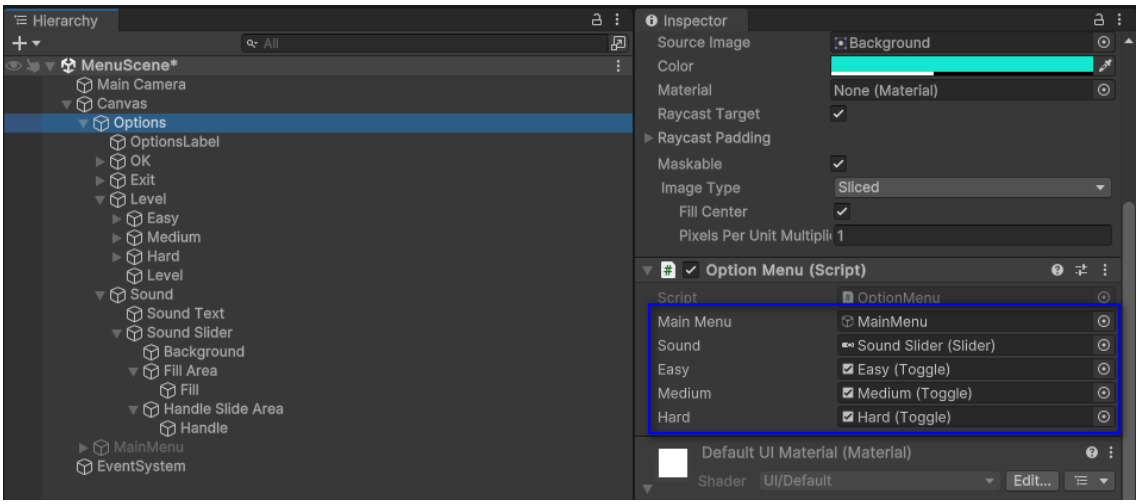
```
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using UnityEngine;
using UnityEngine.UI;

Unity Script | 0 references
public class OptionMenu : MonoBehaviour
{
    public GameObject mainMenu;
    public Slider sound;
    public Toggle easy;
    public Toggle medium;
    public Toggle hard;
    0 references
    public void BackToMenu()
    {
        gameObject.SetActive(false); // Vô hiệu hóa OptionMenu
        mainMenu.SetActive(true); // Bật hộp thoại MainMenu

        // In ra các giá trị đã chọn
        Debug.Log("Âm thanh vừa chọn là " + sound.value);
        string level = "";
        if (easy.isOn) level = "Bạn chọn cấp độ dễ";
        if (medium.isOn) level = "Bạn chọn cấp độ dễ";
        if (hard.isOn) level = "Bạn chọn cấp độ dễ";
        Debug.Log(level);
    }
    0 references
    public void Cancel() {
        gameObject.SetActive(false); // Vô hiệu hóa OptionMenu
        mainMenu.SetActive(true); // Bật hộp thoại MainMenu
        Debug.Log("Không thay đổi giá trị");
    }
```

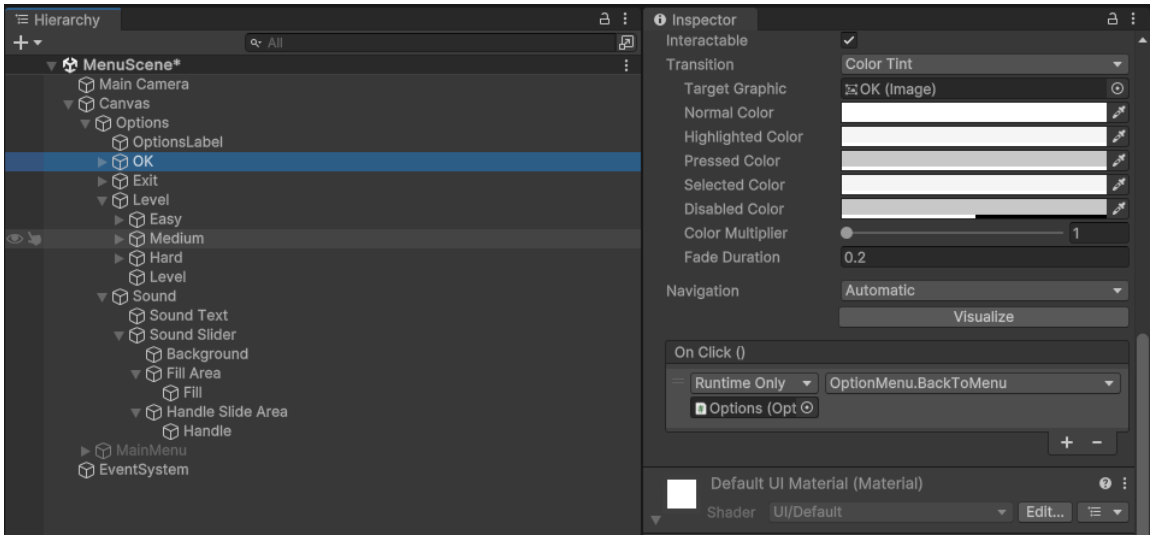
Hình 7.23. Mã nguồn của Option Menu

Gắn mã nguồn vào làm thành phần của Options, kéo các đối tượng tương ứng vào các ô như MainMenu, Sound Slider, Toogles… (Hình 7.24)



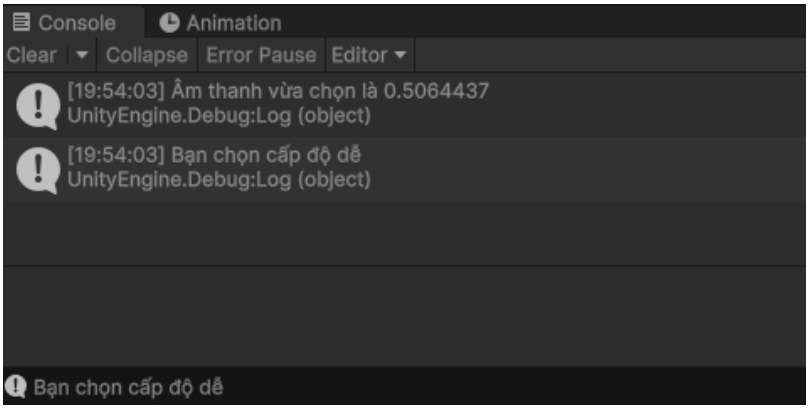
Hình 7.24. Thêm thành phần script Option Menu vào Options và gán các đối tượng tương ứng

Trong Button OK của Option, tạo sự kiện On Click() rồi gắn đối tượng Game và gọi hàm tương ứng như trong Hình 7.25.



Hình 7.25. Gắn sự kiện và gọi hàm cho nút OK

Thực hiện tương tự với nút Cancel, chạy và xem thành quả khi chọn nút OK và Cancel (Hình 7.26).



Hình 7.26. Kết quả khi nhấn nút OK

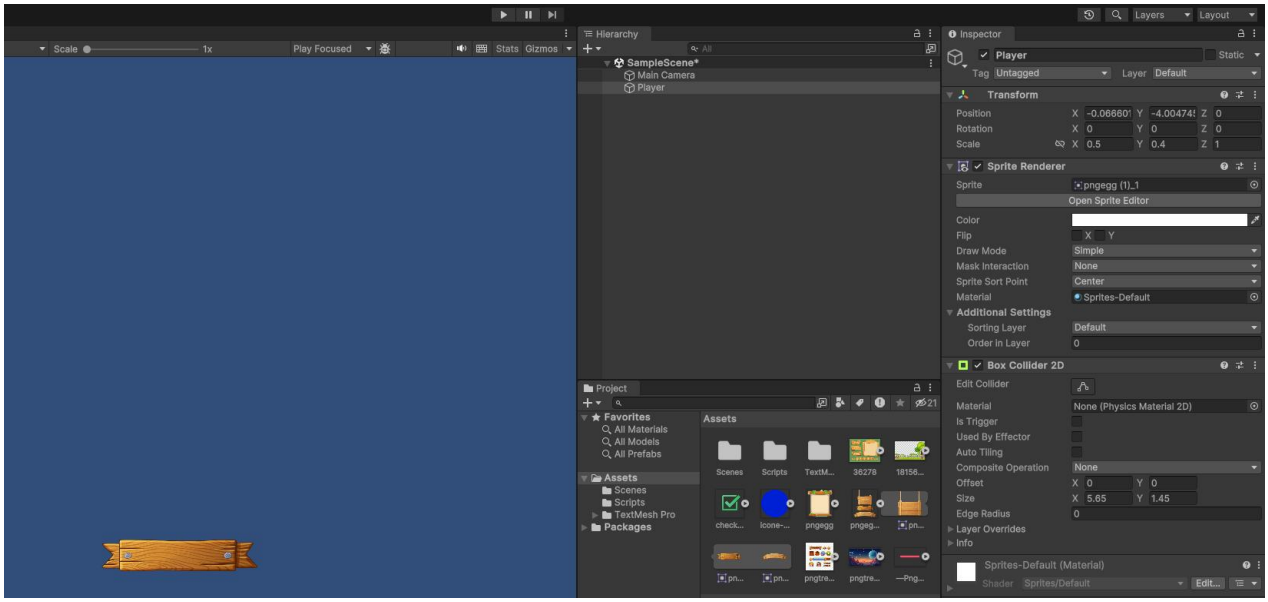
Lưu ý, ở đây chỉ minh họa cách chọn âm thanh, giá trị được in ra tại Console, nếu muốn thay đổi âm thanh thực sự thì cần phải thêm một Audio Source, sau đó điều khiển âm thanh bằng thanh trượt và lấy giá trị gán cho Audio Source.

6. Quản lý điểm số trên màn hình Game

Điểm số của Game thường được hiển thị trực tiếp lên màn hình chính của Game bằng đối tượng Label. Để dễ hiểu, trong phần này sẽ hướng dẫn người dùng xây dựng một Game đơn giản có hiển thị điểm số lên màn hình. Quay lại với Simple Scene ở ví dụ trên (sẽ được chạy khi người dùng nhấn vào nút New Game trên Menu chính).

Trong ví dụ này, sẽ có một thanh trượt đóng vai trò Player nằm ở dưới, một số đối tượng sẽ rơi ngẫu nhiên từ trên xuống, người dùng có nhiệm vụ di chuyển thanh trượt qua lại để hứng đối tượng rơi, nếu đối tượng là bông hoa thì sẽ được cộng điểm, nếu đối tượng là con sâu thì sẽ trừ bớt trên thanh sức khỏe. Trong phần này sẽ hướng dẫn xây dựng phần hiển thị điểm số trên màn hình trước.

Bước 1: Trong Simple Scene, tạo thanh trượt nằm ở dưới màn hình như Hình 7.27. Để thanh trượt này có thể va chạm với vật thể, cần thêm BoxCollider2D cho đối tượng.



Hình 7.27. Thêm vật thể vào trong Simple Scene

Mã nguồn gắn cho đối tượng Player như trong Hình 7.28.

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

Unity Script (1 asset reference) | 0 references
public class PlayerController : MonoBehaviour
{
    private float speed = 10.0f; // Tốc độ dịch chuyển
    Unity Message | 0 references
    void Update()
    {
        // Lấy giá trị x, y khi người dùng nhấn phím di chuyển
        float x = Input.GetAxis("Horizontal");
        float y = Input.GetAxis("Vertical");
        // Dịch chuyển đối tượng theo tốc độ cho trước
        transform.Translate(new Vector2(x, y) * speed * Time.deltaTime);
    }
}
```

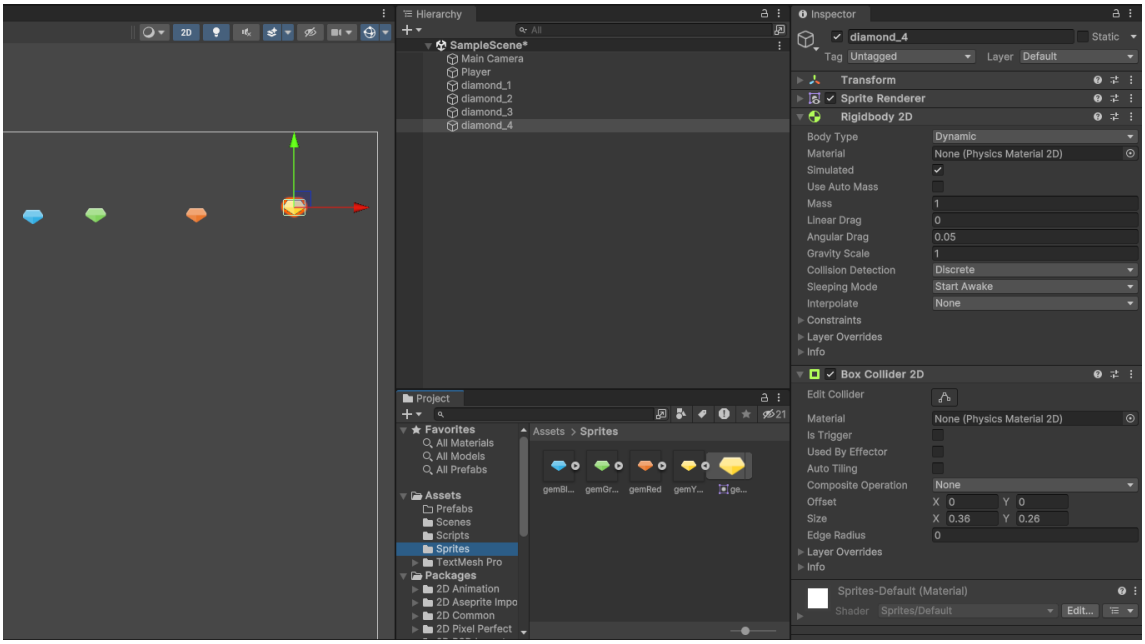
Hình 7.28. Mã nguồn điều khiển Player

Bước 2: Tạo để các đối tượng rơi xuống ngẫu nhiên

Giả sử có bốn loại đối tượng (kim cương) như Hình 7.29 (các đối tượng này có thể tìm thấy dễ dàng trên các trang web chuyên cung cấp các Sprite)

Hình 7.29. Các đối tượng rơi xuống

Sau khi tải các đối tượng này về, đưa vào trong Project của Unity, chuyển về dạng Texture Sprite (2D and UI), kéo vào trong Scene, đặt tên là diamond_1, diamond_2, diamond_3, diamond_4, gắn Rigidbody 2D và Box Collider 2D cho từng đối tượng (Hình 7.30), chạy thử Game, sẽ thấy các đối tượng này rơi xuống. Kéo các 4 đối tượng đó vào Prefab, và xóa trên Scene.



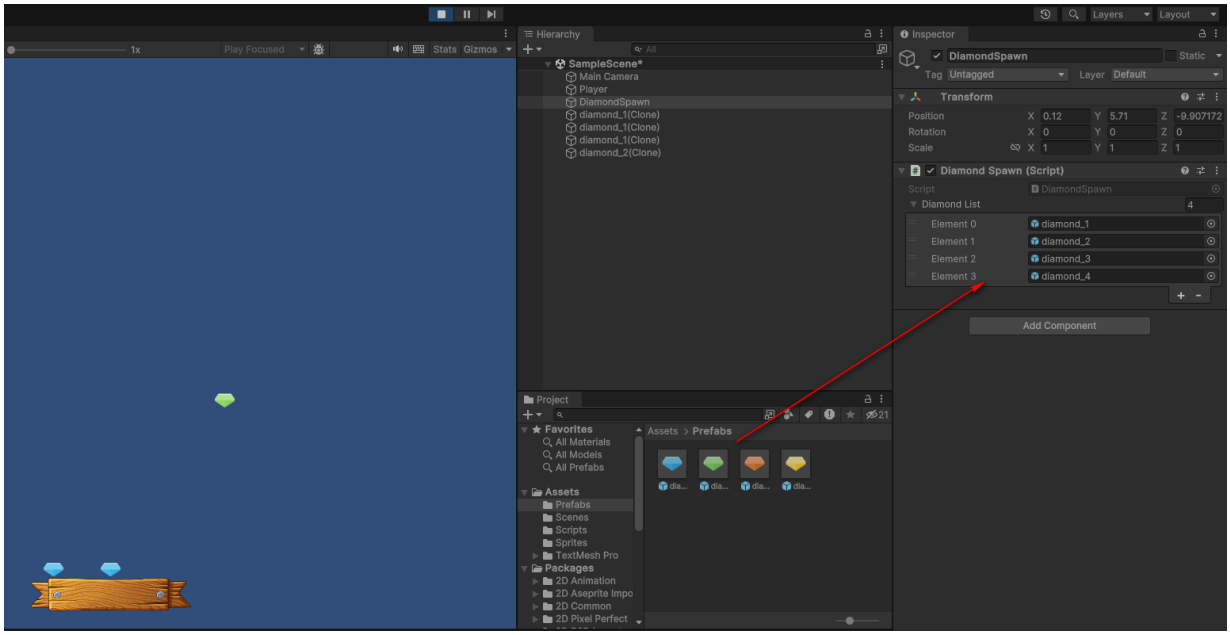
Hình 7.30. Tạo các đối tượng để đưa vào Prefab

Tạo mới một Empty Object đặt tên DiamondSpawn, kéo lên phía trên màn hình, đối tượng này sẽ được lấy giá trị của trục x để làm căn cứ để 4 viên kim cương rơi xuống. Gắn vào một script có tên DiamondSpawn.cs, mã nguồn viết như Hình 7.31.

```
Unity Script | 0 references
public class DiamondSpawn : MonoBehaviour
{
    // Khai báo một mảng chứa các đối tượng kim cương
    public List<GameObject> diamondList;
    // Khai báo đối tượng dùng để tạo số ngẫu nhiên
    System.Random rd = new System.Random();
    Unity Message | 0 references
    void Start()
    {
        // Gọi hàm Spawn() sau 2.5 giây và lặp lại sau 1.5 giây
        InvokeRepeating("Spawn", 2.5f, 1.5f);
    }
    // Hàm Spawn() dùng để phát sinh các viên kim cương
    0 references
    private void Spawn()
    {
        // Tọa độ ngẫu nhiên của kim cương theo chiều x
        float x = rd.Next(-6, 6);
        float y = gameObject.transform.position.y; // y lấy theo DiamondSpawn
        int diamond = rd.Next(0, diamondList.Count);// Lấy ngẫu nhiên kim cương nào
        // Hàm phát sinh ngẫu nhiên viên kim cương tại vị trí ngẫu nhiên x
        Instantiate(diamondList[diamond], new Vector3(x, y, 0), Quaternion.identity);
    }
}
```

Hình 7.31. Mã nguồn phát sinh ngẫu nhiên các viên kim cương tại các vị trí ngẫu nhiên

Quay ra màn hình chính, trong thành phần Script Diamond Spawn, lần lượt tạo các phần tử của mảng DiamondList, kéo các viên kim cương trong Prefabs vào lần lượt, chạy và xem kết quả (Hình 7.32).



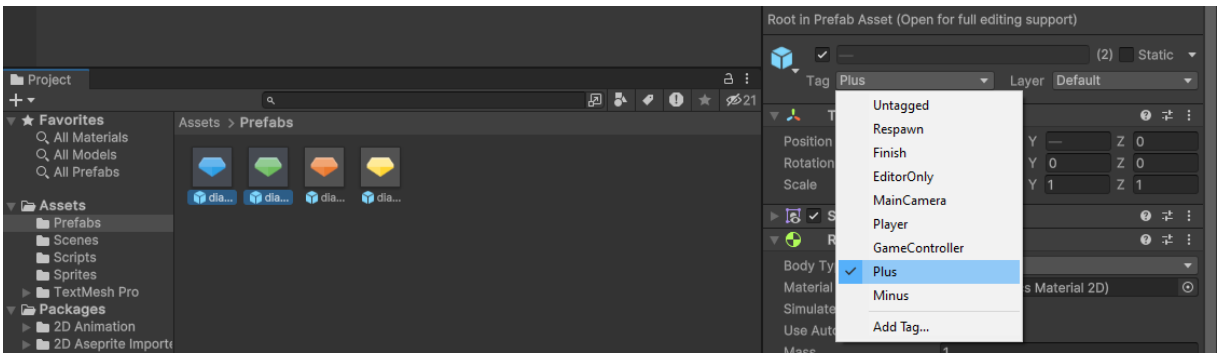
Hình 7.32. Kéo các đối tượng vào trong danh sách, chạy và xem kết quả

Một lưu ý là, cần phải viết một script cho đối tượng viên kim cương để viên kim cương sẽ biến mất khi ra khỏi màn hình. Lệnh này đơn giản nên độc giả có thể tự viết.

Bước 3: Hiển thị điểm số

Có 4 viên kim cương, giả sử rằng máng gỗ hứng được các viên màu xanh là được cộng điểm, còn hứng viên màu vàng và cam là mất điểm. Bước này sẽ cho phép hiển thị điểm số lên màn hình khi hứng được viên màu xanh.

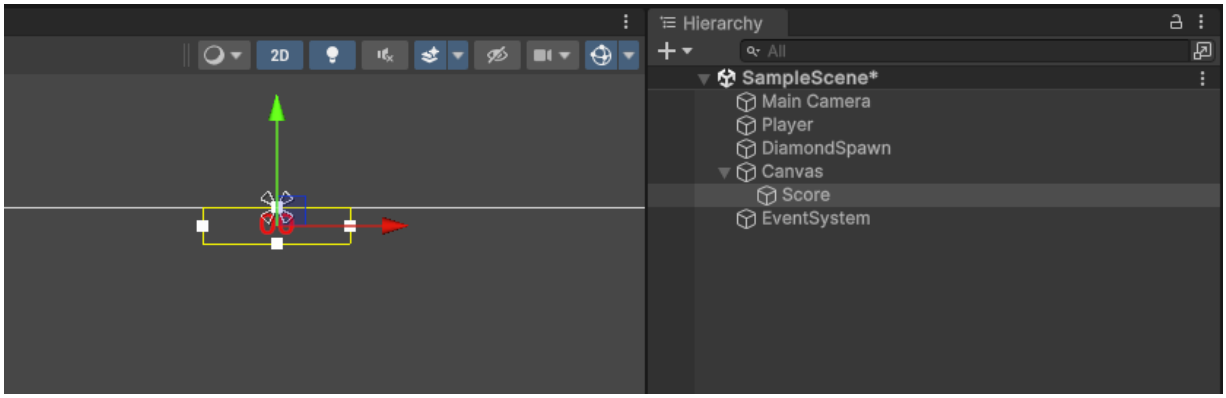
Đầu tiên, cần gán Tag cho các đối tượng viên kim cương trong Prefabs, có hai thẻ cần được gán là Plus (cộng điểm) và Minus (trừ điểm) (Hình 7.33).



Hình 7.33. Gắn thẻ Plus và Minus cho các viên kim cương

Lưu ý là hai thẻ Plus và Minus không có sẵn, cần phải được tạo ra bằng cách chọn Add Tag… trong thuộc tính Tag của các viên kim cương.

Tạo một đối tượng Text – TextMeshPro, đặt tên Score và cho lên nằm giữa màn hình như Hình 7.34. Thay đổi các thuộc tính như đậm nhạt, màu sắc, căn chỉnh cho phù hợp.



Hình 7.34. Thêm thuộc tính Text – TextMeshPro

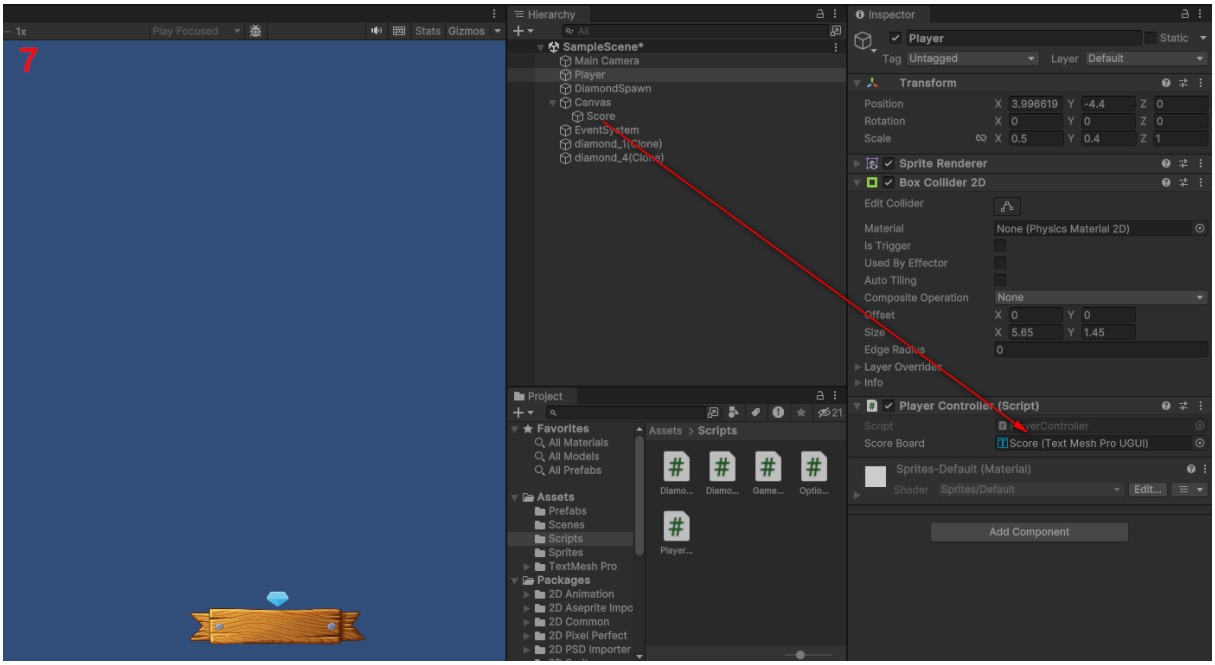
Quay lại Script PlayerController.cs, viết thêm các phương thức và thuộc tính như trong Hình 7.35 để xử lý điểm số và hiển thị lên màn hình.

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

Unity Script (1 asset reference) | 0 references
public class PlayerController : MonoBehaviour
{
    private float speed = 10.0f; // Tốc độ dịch chuyển
    // Đối tượng Text để hiển thị điểm
    public TextMeshProUGUI scoreBoard;
    private int score = 0; // Điểm số
    Unity Message | 0 references
    void Update()
    {
        // Lấy giá trị x, y khi người dùng nhấn phím di chuyển
        float x = Input.GetAxis("Horizontal");
        float y = Input.GetAxis("Vertical");
        // Dịch chuyển đối tượng theo tốc độ cho trước
        transform.Translate(new Vector2(x, y) * speed * Time.deltaTime);
    }
    Unity Message | 0 references
    private void OnCollisionEnter2D(Collision2D collision)
    {
        // Khi va chạm với đội tượng có thẻ là Plus
        if (collision.gameObject.tag == "Plus")
        {
            score += 1; // Tăng điểm lên
            scoreBoard.text = score.ToString(); // Hiển thị
        }
    }
}
```

Hình 7.35. Mã nguồn để xử lý va chạm và hiển thị điểm số

Quay ra màn hình Scene, chọn đối tượng Player, kéo đối tượng Score vào trong thuộc tính Score Board của Scrip, chạy và xem kết quả (Hình 7.36).



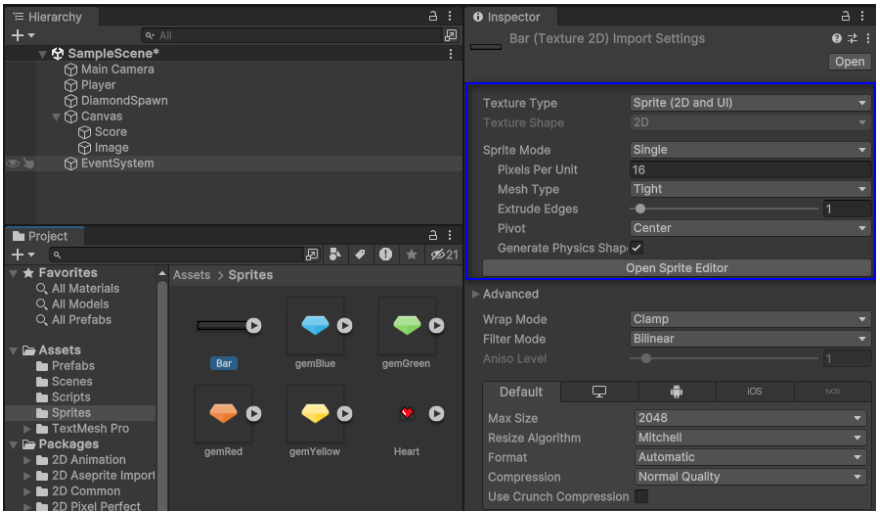
Hình 7.36. Hiển thị điểm số khi chơi

7. Xây dựng thanh sức khỏe cho Player

Để quản lý được sức khỏe cho nhân vật, người ta thường xây dựng một thanh máu để quản lý. Cách xây dựng đó là kết hợp hình ảnh và thanh trượt Slider, đồng thời viết mã nguồn để quản lý. Các bước sau đây sẽ hướng dẫn tạo một thanh sức khỏe để mỗi khi người hứng kim cương ở Mục 6 hứng phải kim cương màu vàng (Minus) thì sẽ bị trừ điểm.

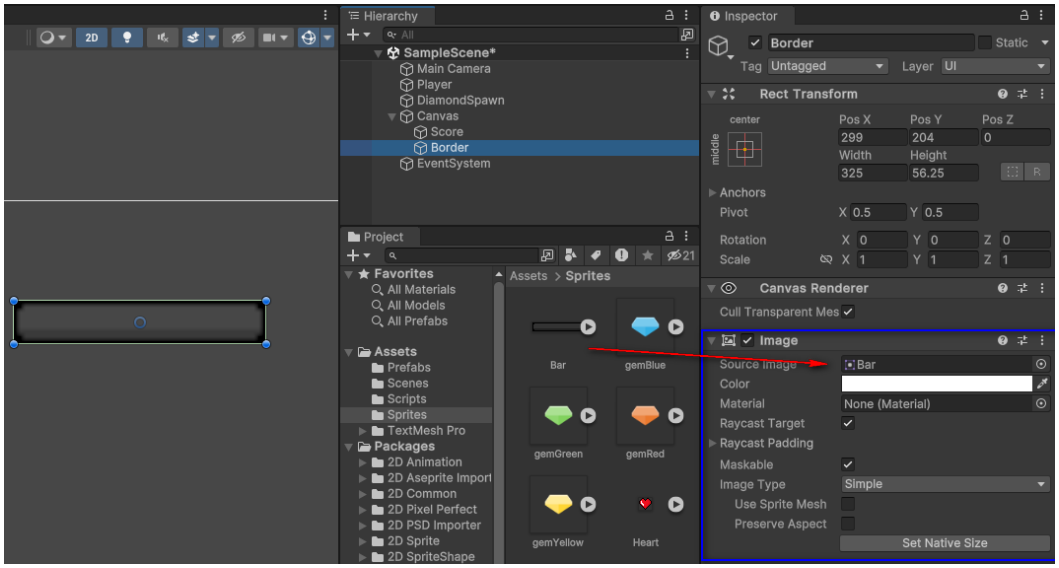
Bước 1: Tìm một hình ảnh để làm Sprite nền cho thanh máu, hình ảnh có thể tìm kiếm dễ dàng với từ khóa “healthy bar sprite + png”. Hướng dẫn này sử dụng tài nguyên được tải về tại: https://github.com/Brackeys/Health-Bar

Sau khi có hình ảnh, chuyển các ảnh vào mục Sprites của Project, điều chỉnh tham số Pixels Per Unit về 16 và Sprite Mode là Single.



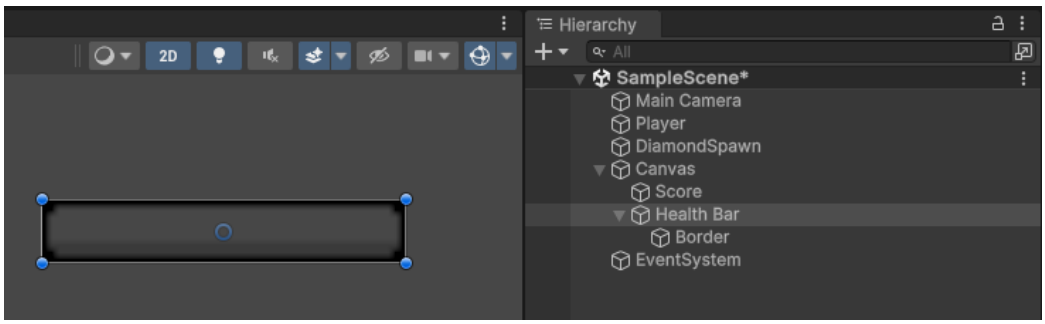
Hình 7.37. Tài nguyên để sử dụng cho thanh sức khỏe

Bước 2: Trong Hierachy, trong đối tượng Canvas, tạo them một đối tượng Image, đặt tên là Border, trên Inspector, kéo Sprite Bar vào trong phần Source Image, nhấn Set Native Size để đối tượng Image có kích thước như thành phần ảnh (Hình 7.38).



Hình 7.37. Tạo Image làm viền cho thanh sức khỏe

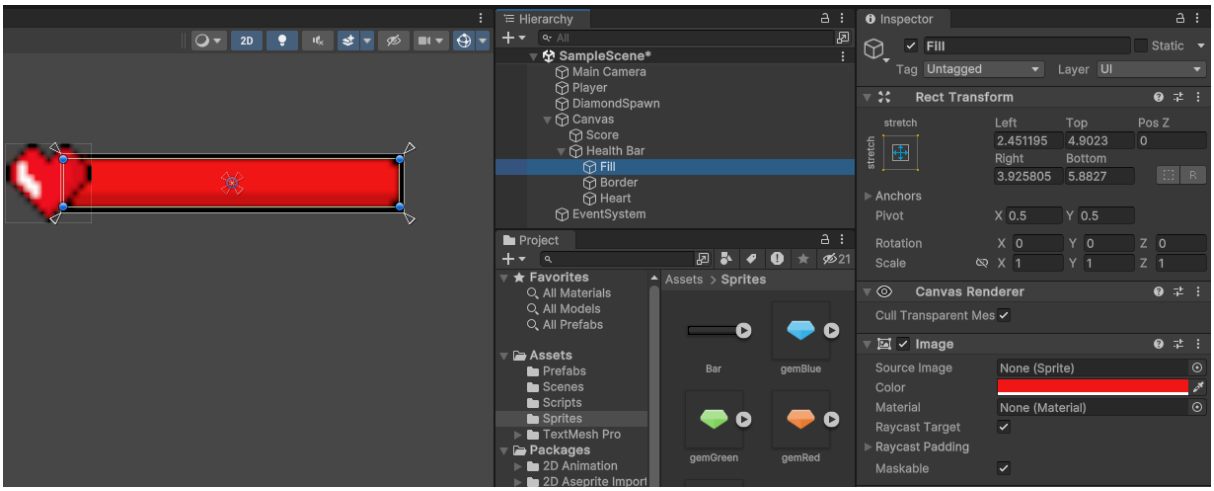
Bước 3: Thêm vào một đối tượng là Game Empty đặt tên là Health Bar, điều chỉnh để kích thước của Health Bar bằng với kích thước của Border, sau đó kéo để Border là con của Health Bar như Hình 7.38.



Hình 7.38. Thêm một đối tượng Healthy Bar

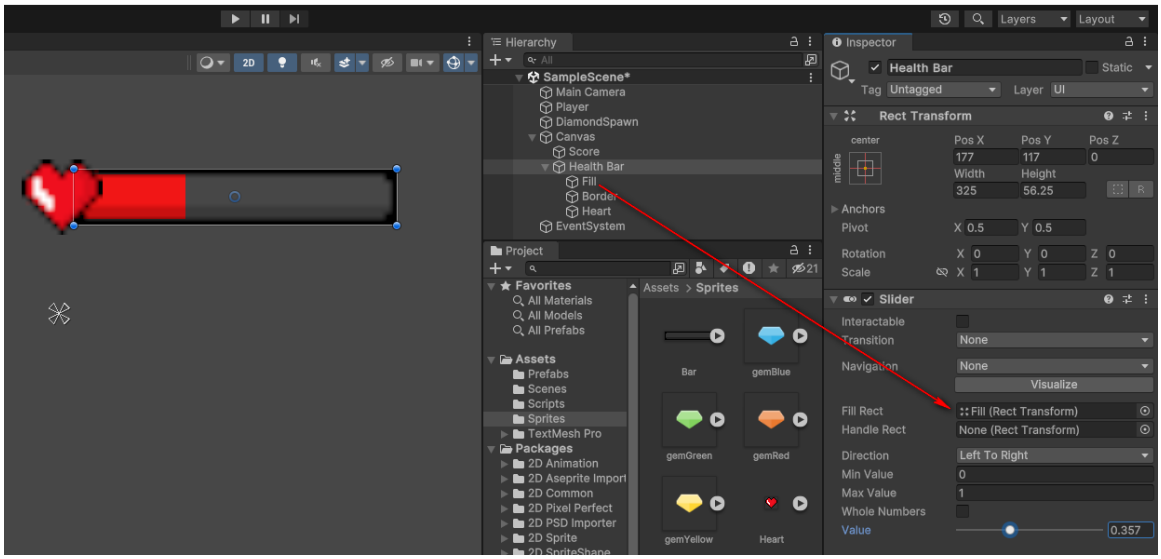
Bước 4: Tạo một Image là con của Health Bar, đặt tên là Fill, điều chỉnh để đối tượng Fill nằm hoàn toàn trong đối tượng Health Bar. Tạo thêm một hình ảnh khác, đặt tên là Heart, kéo sprite Heart vào và nhấn Set Native Size.

Điều chỉnh màu sắc để màu nền của thanh trượt trùng màu nền của hình trái tim như trong Hình 7.39.



Hình 7.39. Thêm hai Image làm hình nền và trang trí cho thanh sức khỏe

Bước 5: Chọn đối tượng Health Bar, trong phần Inspector, thêm vào một thành phần có tên Slider, bỏ chọn Interactable, thuộc tính Transition và Navigation đều để None. Kéo đối tượng Fill vào thuộc tính Fill Rect. Kéo thanh trượt Value để thấy kết quả như Hình 7.40.



Hình 7.40. Thêm thành phần Slider cho Health Bar

Lưu ý: với thành phần Slider, người dung có thể thiết lập để trượt từ trái qua phải hoặc từ phải qua trái. Ngoài ra, có thể thiết lập phần Min Value với giá trị nhỏ nhất (mặc định là 0) và Max Value là giá trị lớn nhất của thanh trượt.

Bước 6: Tạo một mã nguồn có tên HealthBar.cs để gán cho đối tượng Health Bar, để xử lý chức năng của thanh trượt. Mã nguồn này có các mã lệnh như trong Hình 7.41. Lưu ý, cần gọi thư viện UnityEngine.UI để khai báo được đối tượng Slider.

Sau khi gán mã nguồn cho Health Bar, trên thanh Inspector kéo thành phần Slider vào cho thuộc tính Slider của HealthBar.

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

Unity Script | 1 reference
public class HealthBar : MonoBehaviour
{
    // Khai báo một đối tượng Slider
    public Slider slider;

    // Hàm dùng để thiết lập giá trị lớn nhất và giá trị hiện tại cho Slider
    1 reference
    public void SetMaxHealth(int health)
    {
        slider.maxValue = health;
        slider.value = health;
    }
    // Hàm dùng để thiết lập giá trị sức khỏe tại một thời điểm
    1 reference
    public void SetHealth(int health)
    {
        slider.value = health;
    }
}
```

Hình 7.41. Mã nguồn của lớp HealthBar

Bước 7: Trong lớp PalyerController, khai báo các thuộc tính như maxHealth, currentHealth, HealthBar, khởi tạo tham số trong hàm Start(), viết thêm hàm TakeDamage() để mỗi lần bị trừ điểm thì cập nhật tham số và trạng thái của thanh Health Bar như trong Hình 4.2.

```
public int maxHealth = 100; // Sức khỏe tối đa
public int currentHealth = 0; // Sức khỏe hiện tại
public HealthBar healthBar; // Thanh sức khỏe
Unity Message | 0 references
private void Start()
{
    currentHealth = maxHealth;
    healthBar.SetMaxHealth(maxHealth);
}
// Hàm xử lý trừ điểm khi va chạm và cập nhật lên Health Bar
1 reference
void TakeDamage(int damage)
{
    currentHealth -= damage;
    healthBar.SetHealth(currentHealth);
}
```

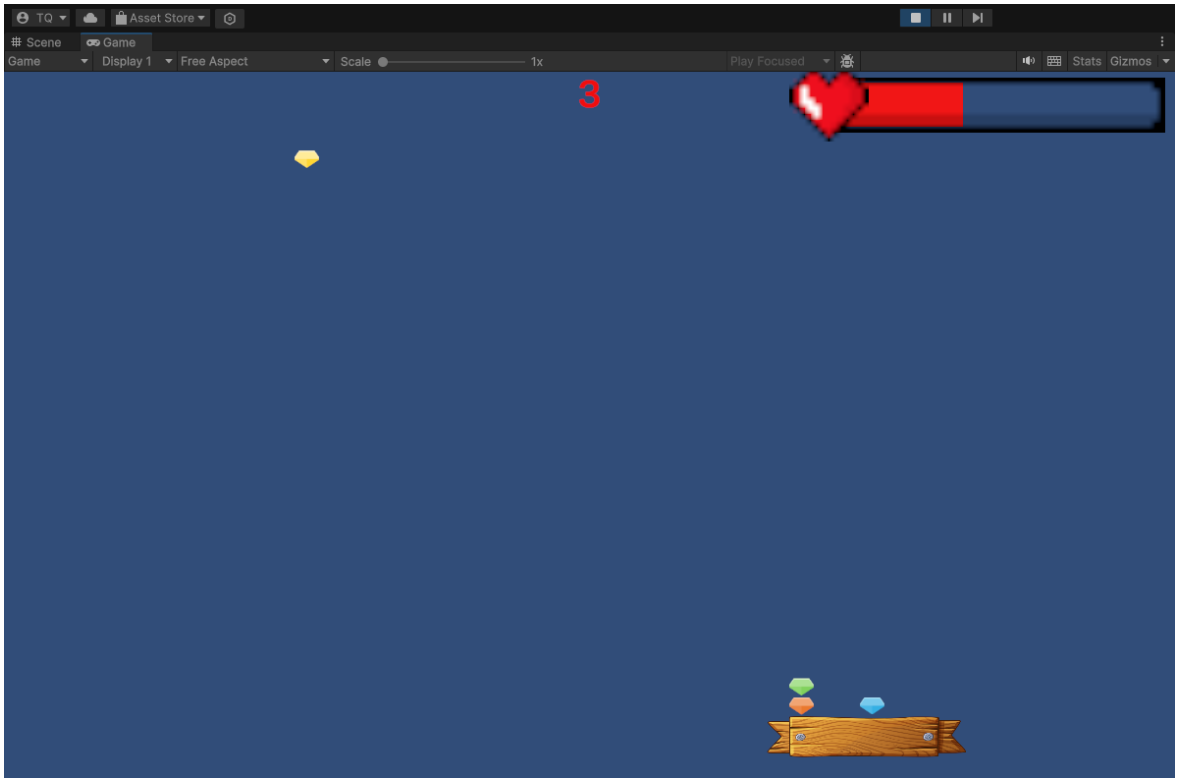
Hình 7.42. Các tham số trong lớp PlayerController

Bước 8: Cũng trong lớp PlayerController, trong hàm OnCollisionEnter2D(), xử lý mã nguồn như sau để khi va chạm với các viên kim cương màu cam và vàng (có gắn thẻ là Minus) thì sẽ bị trừ điểm và thanh sức khỏe sẽ giảm màu (Hình 7.43).

```
private void OnCollisionEnter2D(Collision2D collision)
{
    // Khi va chạm với đội tượng có thẻ là Plus
    if (collision.gameObject.tag == "Plus")
    {
        score += 1; // Tăng điểm lên
        scoreBoard.text = score.ToString(); // Hiển thị
    }
    else if (collision.gameObject.tag == "Minus")
    {
        TakeDamage(20);
    }
}
```

Hình 7.43. Mã nguồn khi va chạm với viên kim cương gắn thẻ là Minus.

Qua Unity, chọn Player, trên Inspector, kéo đối tượng Health Ba vào trong thuộc tính Health Bar của thành phần Player Controller. Chạy và xem kết quả: Khi game hứng phải viên kim cương màu cam và vàng thì sẽ bị trừ thanh máu (sức khỏe bị giảm) (Hình 7.44).



Hình 7.44. Game hoàn thành khi chạy

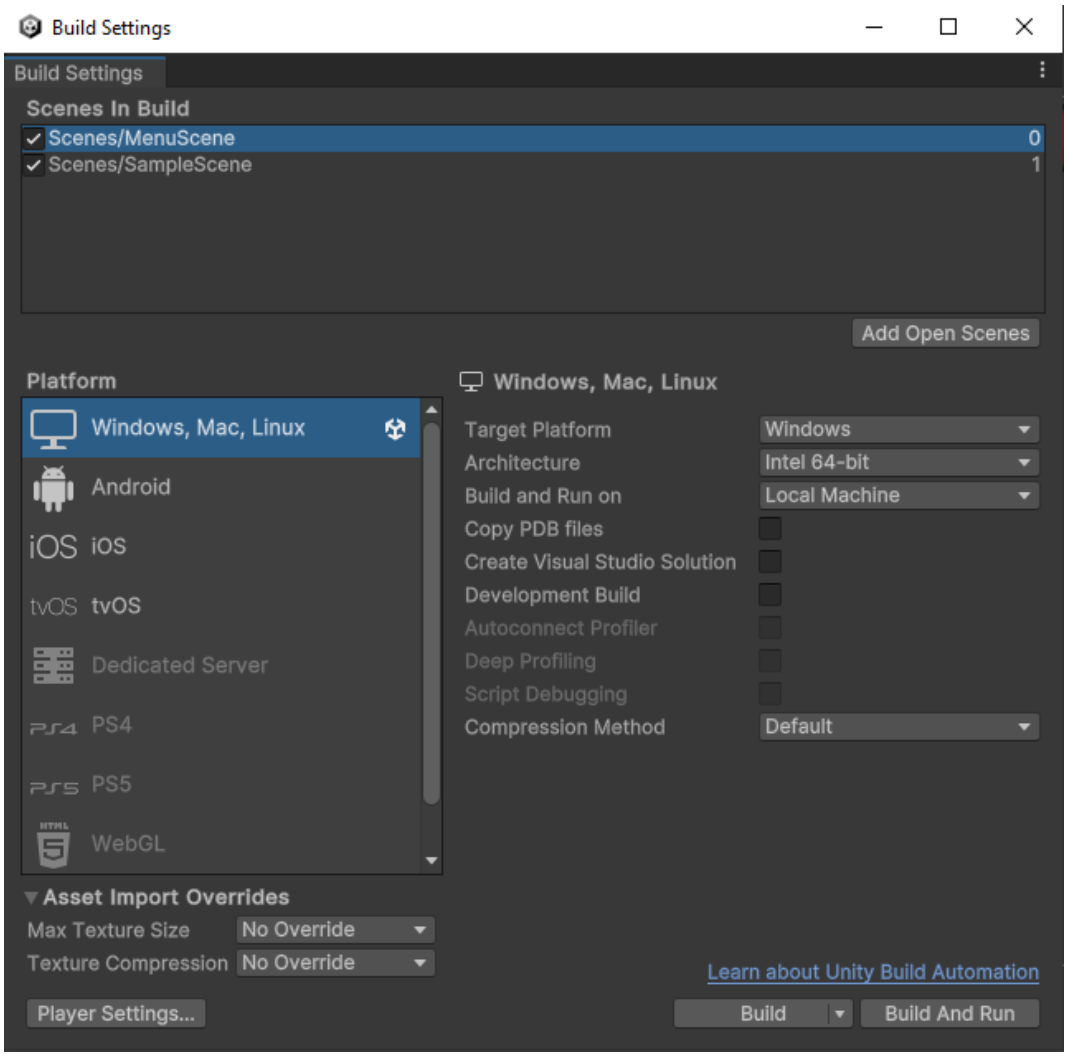
8. Kết chương

Chương 8. Xuất bản ứng dụng

Unity là một ứng dụng đa nền tảng, nghĩa là khi xây dựng xong thì có thể phát hành thành các sản phẩm chạy được trên nhiều môi trường khác nhau. Các nền tảng mà Unity hỗ trợ chạy và thực thi bao gồm Window (.exe), iOS, Web… Chương này sẽ mô tả một số phương pháp xuất ứng dụng, các thiết lập cài đặt và cách nhúng Google Admob (quảng cáo) vào trong ứng dụng.

1. Xuất bản ứng dụng

Để xuất bản được một ứng dụng hoàn chỉnh thì việc đầu tiên cần làm đó là cấu hình một số tham số của dự án trước khi xuất bản. Unity cho phép cấu hình tham số bằng cách vào File, chọn Build Settings (lưu ý rằng, có thể chọn chức năng Build And Run nếu như các tham số đ0ã được thiết lập). Một hộp thoại hiện ra như trong Hình 8.1.



Hình 8.1. Hộp thoại Build Settings

Hộp thoại này có một số chức năng cơ bản như sau:

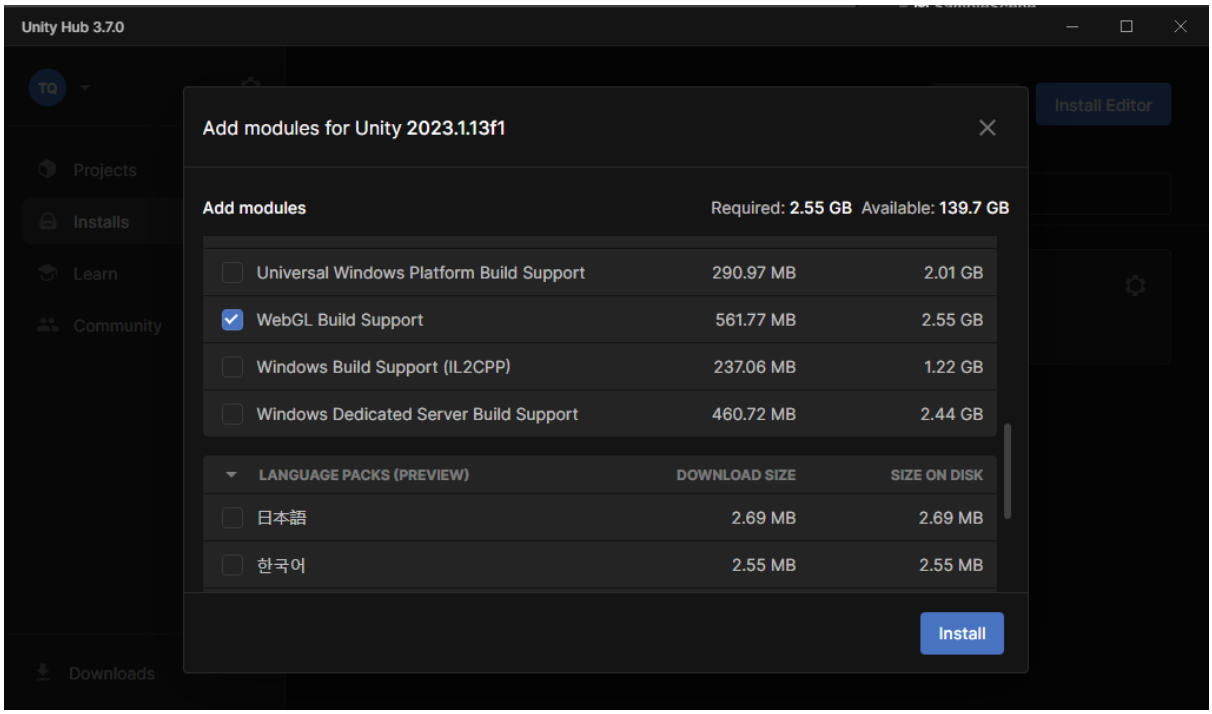
- **Scenes In Build**: Chức năng này cho phép thêm các màn hiện có trong Project, các màn sẽ được đánh số từ 0 theo tên của Scene. Thứ tự này góp phần quan trọng trong hàm LoadScene() được đề cập trong Chương 3. Nút Add Open Scene cho phép thêm vào Scene hiện đang được mở.

- **Platform**: Là nền tảng phát hành. Unity hỗ trợ rất nhiều nền tảng phát hành như Windows, Mac, Linux, Android, iOS, TV, Play Station, WebNG. Trong phần này giới thiệu 3 nền tảng chính được mô tả như trong Bảng 8.1.

Bảng 8.1. Các nền tảng phát hành trong Unity

Nền tảng	**Giải thích và tham số**
Window, Mac, Linux	Đây là nền tảng phát hành trên Window, khi chọn loại nền tảng trong phần Target Platform, Unity sẽ xuất ra định dạng tập tin phù hợp để chạy trên nền tảng đó. Ví dụ với Windows sẽ xuất bản thành tập tin .exe. Ngoài ra, khi chọn xuất bản với nền tảng này, Unity cũng cho phép chọn kiến trúc đồ họa trong phần Architechure.
Android	Là nền tảng xuất bản để chạy trên hệ điều hành Android. Nền tảng này sẽ xuất bản tập tin apk để thực thi trên Android. Cần lưu ý rằng, màn hình điện thoại di động có kích thước nhỏ nên khi xuất bản trên màn hình này cần phải tùy chỉnh các tham số về màn hình trong phần Player Settings (được đề cập ở mục tiếp theo). Ngoài ra Unity cũng cho phép thiết lập một số tùy chỉnh khác trong Android như ETC, Device…
iOS	Nền tảng xuất bản trên hệ điều hành iOS, cho phép người dùng chọn Xcode để chạy ứng dụng.
tvOS	Cho phép xuất bản ứng dụng chạy được trên hệ điều hành của Tivi

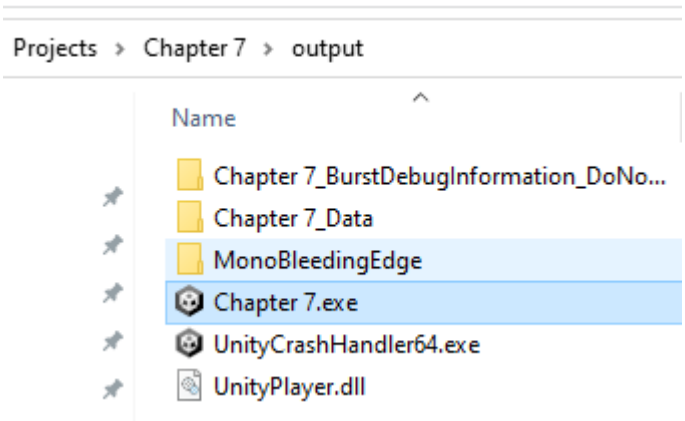
Lưu ý là một số nền tảng trong Hình 8.1 bị mờ đi, do quá trình cài đặt tác giả đã không chọn để giảm dung lượng bộ nhớ. Nếu muốn hỗ trợ trở lại thì cần vào Unity Hub, chọn mục Installs, trong chỗ phiên bản Unity, chọn Settings, chọn Add Modules, chọn nền tảng cần hỗ trợ và bấm Install (Hình 8.2)



Hình 8.2. Cài đặt thêm nền tảng phát triển

Mặc định Unity thiết lập nền tảng Windows, Mac, Linux, nếu muốn chuyển qua các nền tảng còn lại, cần bấm Swich Platform trước khi bấm Build hoặc Buid And Run. Khi đó Unity sẽ yêu cầu chọn thư mục để lưu trữ, chọn thư mục, Unity sẽ trải qua quá trình xuất bản ứng dụng, nếu có lỗi sẽ yêu cầu sửa lỗi trước khi xuất bản.

Nếu không gặp lỗi gì, hệ thống sẽ xuất bản toàn bộ ứng dụng và đóng gói tập tin tương ứng trong một thư mục, người dùng có thể chạy nó (Hình 8.3 và Hình 8.4).



Hình 8.3. Thư mục xuất bản

Hình 8.4. Minh họa game khi chạy bằng tập tin .exe.

Cần phải lưu ý rằng, khi game được xây dựng bằng bản cá nhân miễn phí (Personal) thì mỗi lần chạy Unity sẽ có dòng chữ "Made by Unity", nếu muốn mất dòng chữ này cần phải dùng bản trả phí.

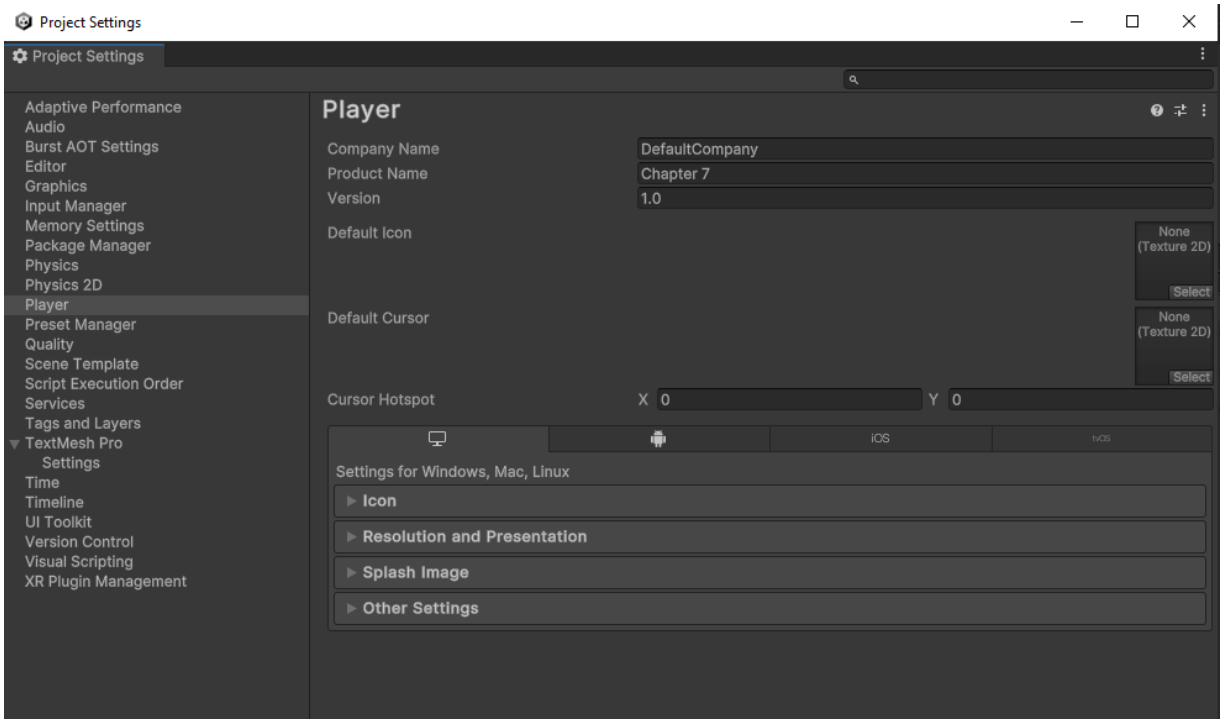
2. Một số thiết lập người dùng

Trong hộp thoại Build Settings có một chức năng cho phép thiết lập một số tham số của người dùng trước khi xuất bản, đó chính là Player Settings, khi bấm nút này thì một hộp thoại mới xuất hiện cho phép tùy chỉnh một số chức năng với người dùng (Hình 8.5).

Các tính năng chính của hộp thoại Player Settings bao gồm một số tùy chỉnh như tên công ty, tên sản phẩm, Icon mặc định, con trỏ mặc định. Tùy vào việc xuất bản trên nền tảng nào mà Unity sẽ cho chọn các thuộc tính của nền tảng đó. Một số nền tảng hỗ trợ trong Windows, Mac, Linux như:

- ***Icon***: Cho phép thiết lập các Icon cho Game;
- ***Resolution and Presentation***: Cho phép thiết lập độ phân giải và cách hiển thị.

- ***Splash Image:*** Thiết lập độ nhạy cho màn hình khi bắt đầu chơi game;
- ***Other Settings:*** Một số thuộc tính khác như màu sắc, đồ họa cho game…



Hình 8.5. Một số tính năng thiết lập người dùng

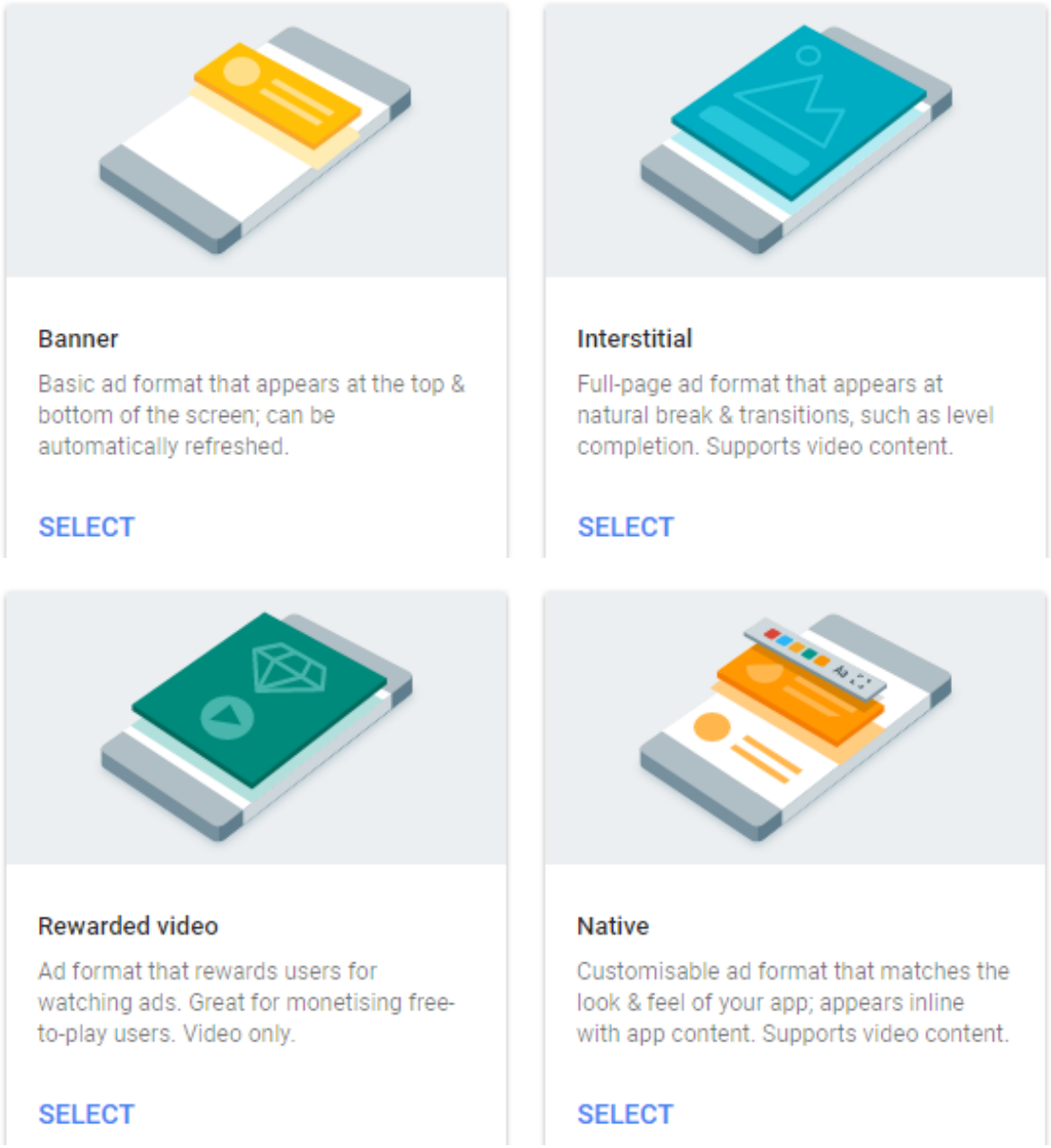
3. Tích hợp quảng cáo Google Admob vào Unity

Khi phát hành game, các nhà phát triển Game thường lồng quảng cáo, đây là chính sách mà các công ty lưu trữ (Store) thực hiện để các doanh nghiệp có thể gắn quảng cáo lên sản phẩm được tải lên kho của họ. Một trong những phương pháp thường được dùng để gắn quảng cáo là sử dụng Google Admob. Google Admob cho phép gắn quảng cáo khi chạy ứng dụng hoặc khi tạm dừng ứng dụng. Doanh thu của người làm ứng dụng sẽ được Google tính toán khi có nhiều lượt click lên quảng cáo của ứng dụng đó.

Để tạo và nhúng được Google Admob vào trong trang, người dùng cần vào Google Admob, tạo tài khoản và tạo một id của ứng dụng. Google Admob cung cấp 4 loại quảng cáo như trong Hình 8.6. Các loại quảng cáo bao gồm:

- ***Banner***: Là loại quảng cáo xuất hiện trên đầu hoặc phía dưới màn hình, loại quảng cáo này có thể tự động làm mới.
- ***Interstial***: Là loại quảng cáo dạng tràn màn hình (Full-page), sẽ hiển thị và chiếm hết màn hình. Loại quảng cáo này có hỗ trợ video.
- ***Rewarded video***: Loại quảng cáo phần thưởng, cho phép hiển thị như là phần thưởng khi người chơi thắng hoặc yêu cầu bấm nút để nhận thêm điểm số. Loại quảng cáo này chỉ hỗ trợ video.
- ***Native***: Loại quảng cáo này sẽ tùy biến và ẩn vào trong sản phẩm một cách tự nhiên.

Mỗi loại quảng cáo sẽ được cấp một ID riêng biệt, người dùng cần hiển thị loại quảng cáo nào thì chọn loại đó để lấy ID. Ngoài ra, quảng cáo dạng Native chỉ hiển thị khi người dùng chạy trực tiếp trên điện thoại, còn các loại quảng cáo khác thì có thể hiển thị khi chạy trong môi trường Unity.



Hình 8.6. Các loại quảng cáo trong Google Admob

Ví dụ sau đây sẽ xây dựng 4 loại quảng cáo trên. Mỗi khi người dùng click vào loại quảng cáo nào thì quảng cáo đó sẽ xuất hiện. Trong ví dụ này, sẽ tạm dùng các ID như sau (lưu ý, người đọc nên tự tạo tài khoản và ID của riêng mình):

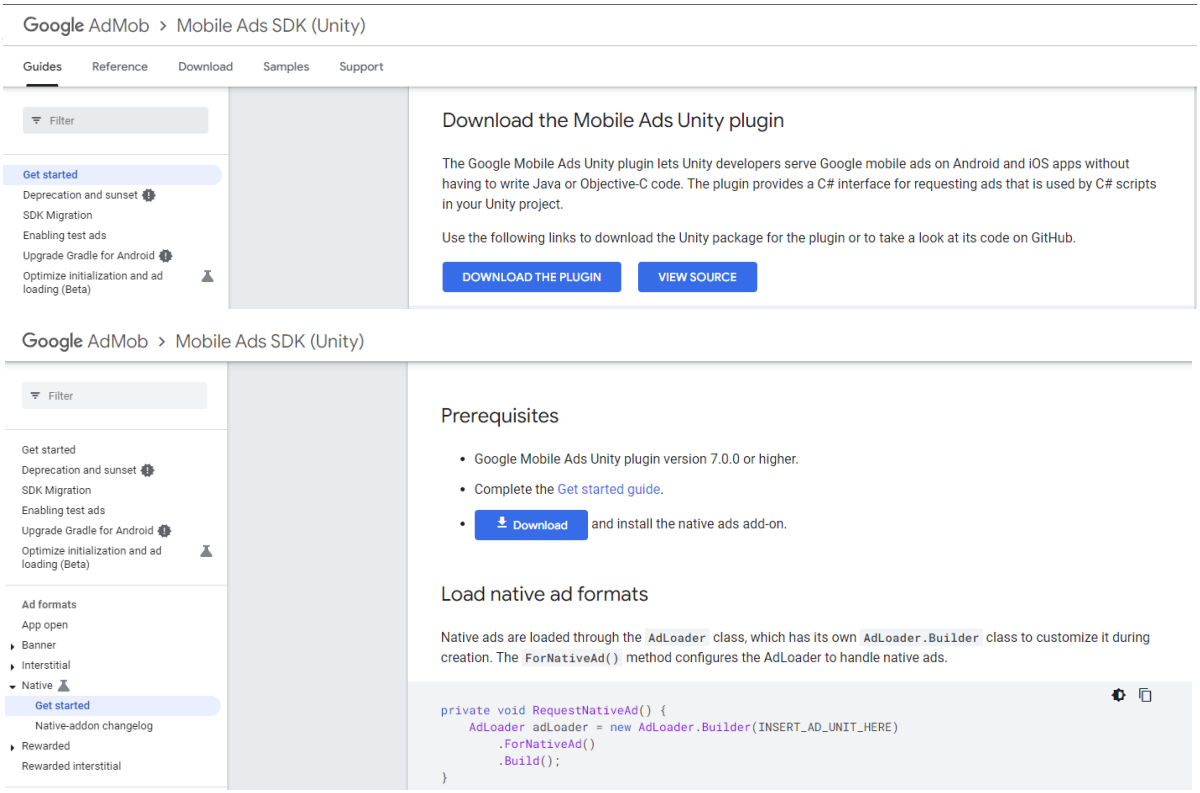
Test 's ID:

app id: ca-app-pub-3940256099942544~3347511713
//android
bannerId ca-app-pub-3940256099942544/6300978111
interid ca-app-pub-3940256099942544/1033173712
rewardedId ca-app-pub-3940256099942544/5224354917
nativeId ca-app-pub-3940256099942544/2247696110
//ios
bannerId ca-app-pub-3940256099942544/2934735716
interid ca-app-pub-3940256099942544/4411468910
rewardedId ca-app-pub-3940256099942544/1712485313
nativeId ca-app-pub-3940256099942544/3986624511

Bước 1: Tạo một Scene mới, xây dựng một giao diện như Hình 8.7. Mục đích là khi người dùng bấm vào nút tương ứng thì sẽ hiển thị các loại quảng cáo tương ứng, quảng cáo dạng Banner thì không thể tắt, các loại quảng cáo khác thì có nút để người dùng tắt khi xong quảng cáo. Riêng quảng cáo Native thì nó không hiện lên trên Unity.

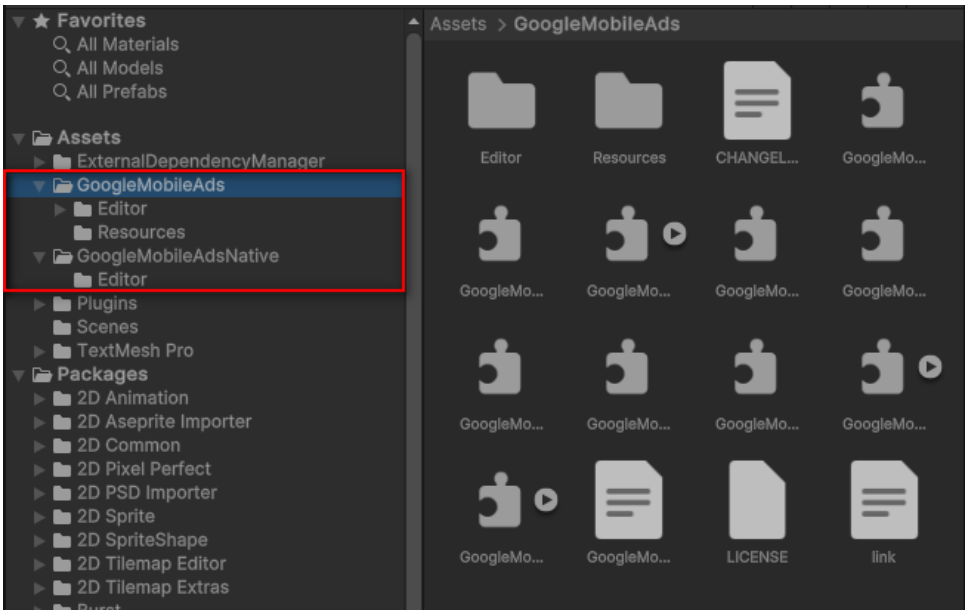
Hình 8.7. Thiết kế giao diện hiển thị các loại Google Admob

Bước 2: Vào trang Google Admob, trong Get stated, tải Mobile Ads Unity plugin như Hình 8.8. Quảng cáo dạng Native cần tải thêm thư viện, nên cũng được tải về tại múc Native\Get stated (Hình 8.8).



Hình 8.8. Tải thư viện Mobile Ads và Mobile Native Ads từ trang Google Admob

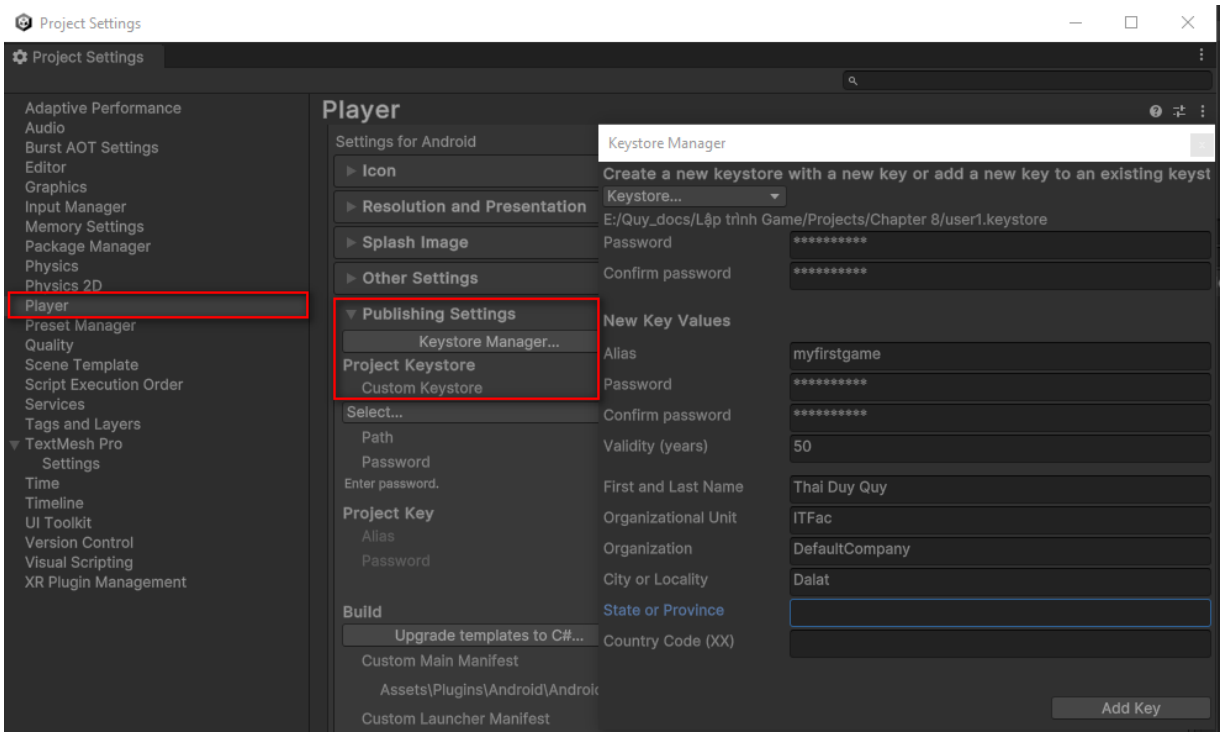
Sau khi tải về, dùng chuột kéo các gói vào trong dự án như trong Hình 8.9



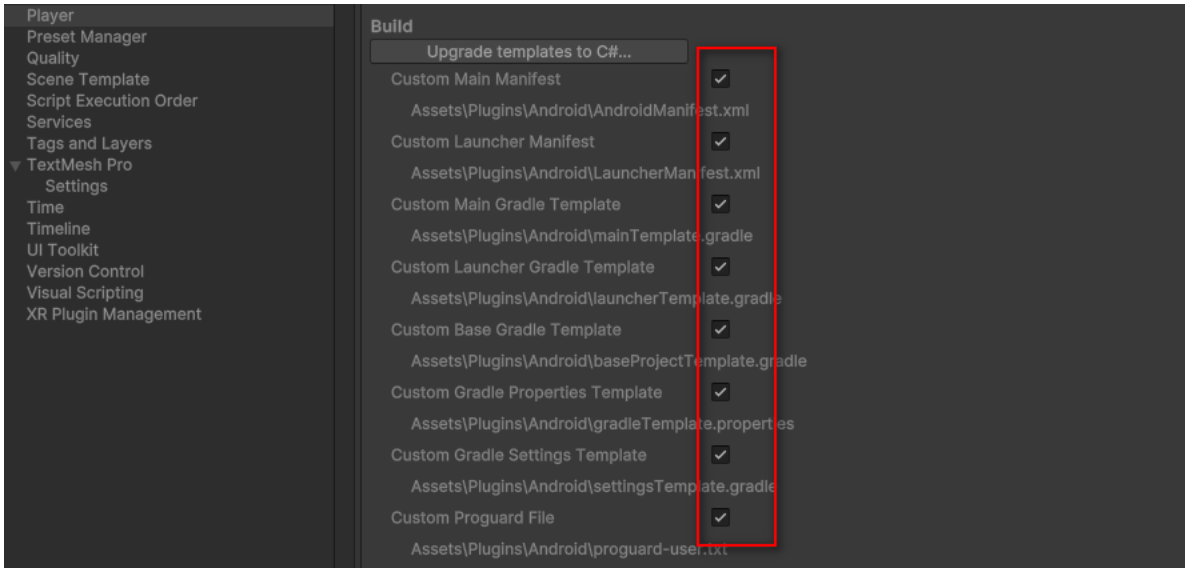
Hình 8.9. Kéo các gói đã tải vào trong dự án

Bước 3: Thiết lập Keystore: Vào Edit, chọn Project Settings, chọn mục Player, chọn phần Publishing Settings, chọn Keystore Manager…

Keystore là một tập tin dạng lưu trữ của hệ thống, cho phép quản lý các tiến trình trong việc nhúng quảng cáo. Khi chọn Keystore Manager… hệ thống yêu cầu nhập thông tin về Keystore, nếu chưa có thì tạo mới bằng cách vào Keystore…, Create New, Anywhere (Nếu đã có thì chỉ cần chọn tập tin và nhập mật khẩu), sau đó nhấn Add key (Hình 8.10). Trong cửa sổ Player Settings, chọn hết các mục trong phần Build như Hình 8.11.

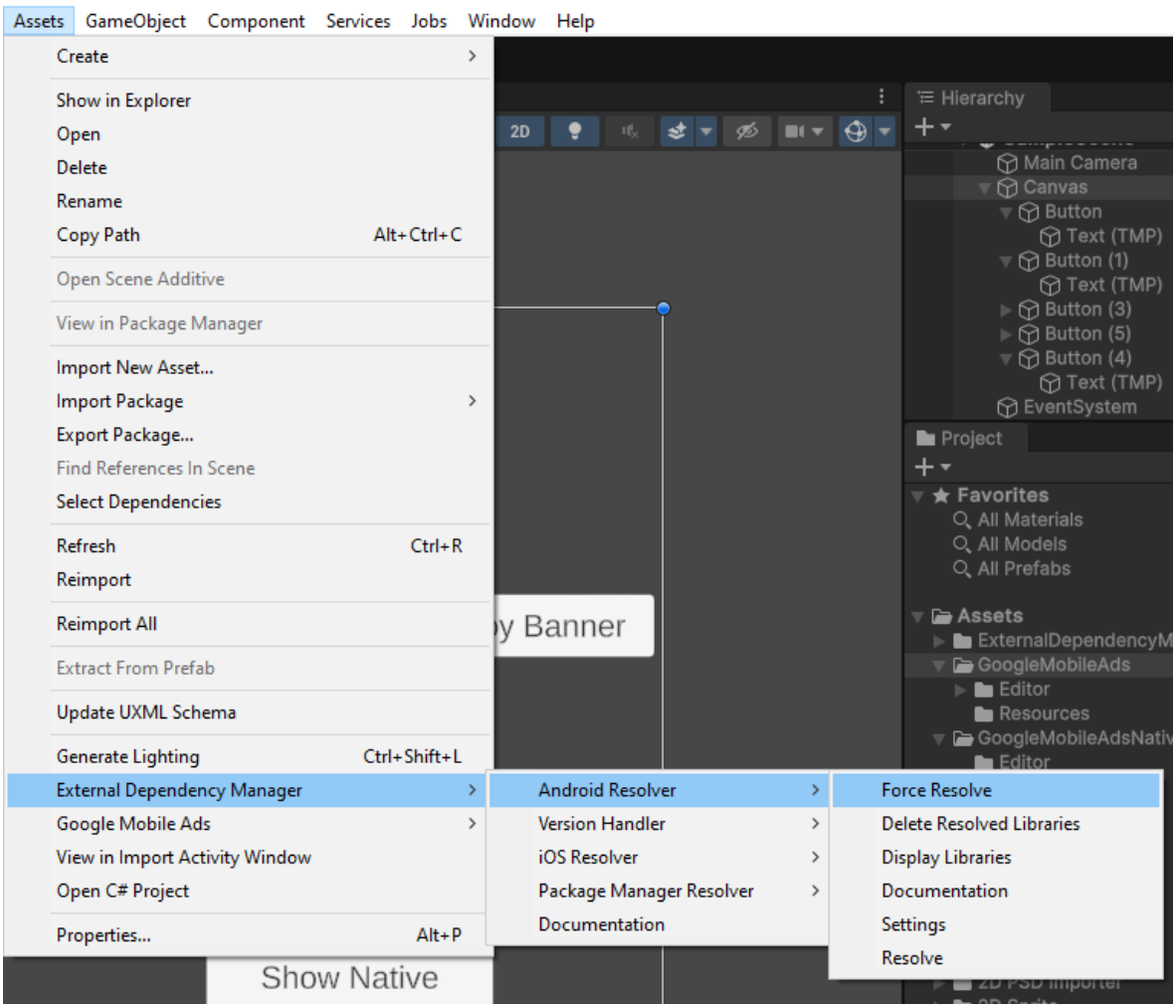


Hình 8.10. Nhập keystore cho quảng cáo



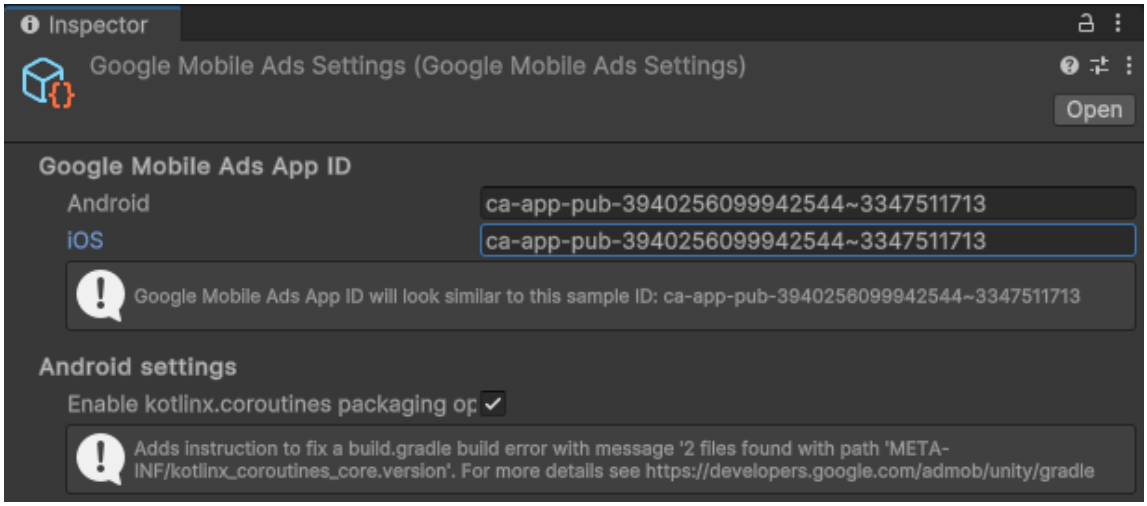
Hình 8.11. Chọn hết các mục sau khi thêm keystore

Bước 4: Vào Assets, chọn External Dependency, chọn Android Resolver, chọn Force Resolver (Hình 8.12) sau đó nhấn OK khi có hộp thoại xuất hiện.



Hình 8.12. Chạy chức năng Force Resolve

Tiếp tục vào Assets, chọn Google Mobile Ads, chọn Settings…, rồi điền các ID của Android và iOS đã cung cấp vào trong Inspector như Hình 8.13.



Hình 8.13. Cấu hình cho Google Mobile Ads

Bước 5: Tạo mới một Empty Object, đặt tên là GoogleAddManager, sau đó tạo một tập tin có tên là AdmobAdsScript.cs, gắn cho đối tượng vừa tạo.

Trong tập tin AdmobAdsScript.cs, khai báo các lệnh như trong Hình 8.14. Lưu ý, các ID là các giá trị đã được cung cấp ở trên. Đừng quên gọi thư viện như GoogleMobileAds.Api.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using GoogleMobileAds.Api;
using System;

public class AdmodAdsScript : MonoBehaviour
{
    // Khai báo đối tượng app ID
    public string appID = "ca-app-pub-3940256099942544~3347511713";

#if UNITY_ANDROID // Các ID dùng cho Android
    string bannerId = "ca-app-pub-3940256099942544/6300978111";
    string interId = "ca-app-pub-3940256099942544/1033173712";
    string rewardedId = "ca-app-pub-3940256099942544/5224354917";
    string nativeId = "ca-app-pub-3940256099942544/2247696110";
#elif UNITY_IPHONE // Các ID dùng cho iOS
    string bannerId = "ca-app-pub-3940256099942544/2934735716";
    string interId = "ca-app-pub-3940256099942544/4411468910";
    string rewardedId = "ca-app-pub-3940256099942544/1712485313";
    string nativeId = "ca-app-pub-3940256099942544/3986624511";
#endif
    // Khai báo các đối tượng là các loại quảng cáo
    BannerView bannerView;
    InterstitialAd interstitialAd;
    RewardedAd rewardedAd;
    NativeAd nativeAd;
```

Hình 8.14. Mã nguồn khai báo đối tượng trong AdmobAdsScript.cs

Trong hàm Start, mã nguồn được viết như trong Hình 8.15.

```
private void Start()
{
    // Khởi tạo các tham số của MobileAds
    MobileAds.RaiseAdEventsOnUnityMainThread = true;
    MobileAds.Initialize(initStatus =>
    {
        Debug.Log("Add initialsed !!");
    });
}
```

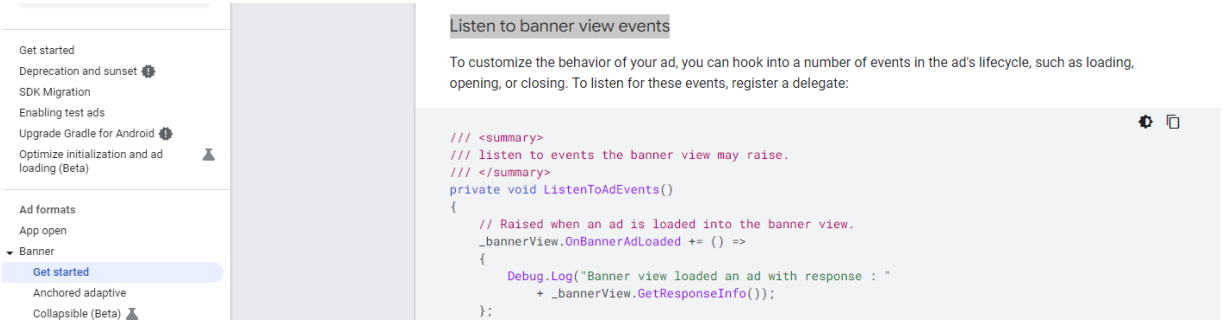
Hình 8.15. Mã nguồn hàm Start()

Bước 6: Để hiển thị banner lên màn hình, viết hai hàm là LoadBannerAdd() và DestroyBannerAd() như Hình 8.16.

```
#region Banner
public void LoadBannerAd()
{
    // Khởi tạo banner
    if (bannerView != null)
        DestroyBannerAd();
    bannerView = new BannerView(bannerId, AdSize.MediumRectangle, AdPosition.Top);
    // lắng nghe các sự kiện
    ListenToAdEvents();
    // Tải banner lên màn hình
    var adRequest = new AdRequest();
    adRequest.Keywords.Add("unity-admod-sample");
    bannerView.LoadAd(adRequest); // hiển thị Banner
}
private void ListenToAdEvents()...
public void DestroyBannerAd()
{
    if(bannerView != null)
    {
        bannerView.Destroy();
        bannerView = null;
    }
}
#endregion
```

Hình 8.16. Mã nguồn để tải phần Banner

Lưu ý, hàm ListenToAdEvents() là hàm dùng để lắng nghe các sự kiện, nội dung hàm này có thể được copy từ trang web của Google Admob, phần Banner, Get started tại mục Listen to banner view events (Hình 8.17). Hàm này cũng có thể để trống nếu muốn.



Hình 8.17. Nội dung hàm ListenToAdEvents

Thực hiện tương tự với LoadInterstitialAd() và DestroyInterstitialAd() như Hình 8.18.

```
#region Interstitial add
public void LoadInterstitialAd()
{
    // Khởi tạo banner
    if (interstitialAd != null)
        DestroyInterstitialAd();
    var adRequest = new AdRequest();
    adRequest.Keywords.Add("unity-admod-sample");
    InterstitialAd.Load(interId, adRequest, (InterstitialAd ad, LoadAdError error) =>
    {
        if(error != null || ad  == null) return;
        interstitialAd = ad;
        // lắng nghe các sự kiện
        InterstitialAdEvent(interstitialAd);
    });

    if (interstitialAd != null && interstitialAd.CanShowAd())
        interstitialAd.Show();
}
private void InterstitialAdEvent(InterstitialAd interstitialAd)...
public void DestroyInterstitialAd()
{
    interstitialAd.Destroy();
    interstitialAd = null;
}
#endregion
```

Hình 8.18. Mã nguồn để tải phần InterstitialAd

Lưu ý, hàm InterstitialAdEvent() cũng có thể được copy từ trang web của Google Admob, phần Interstitial, Get started tại mục Listen to interstitial ad events hoặc để trống.

Thực hiện tương tự với LoadRewardedAd () và DestroyRewardedAd() như Hình 8.19.

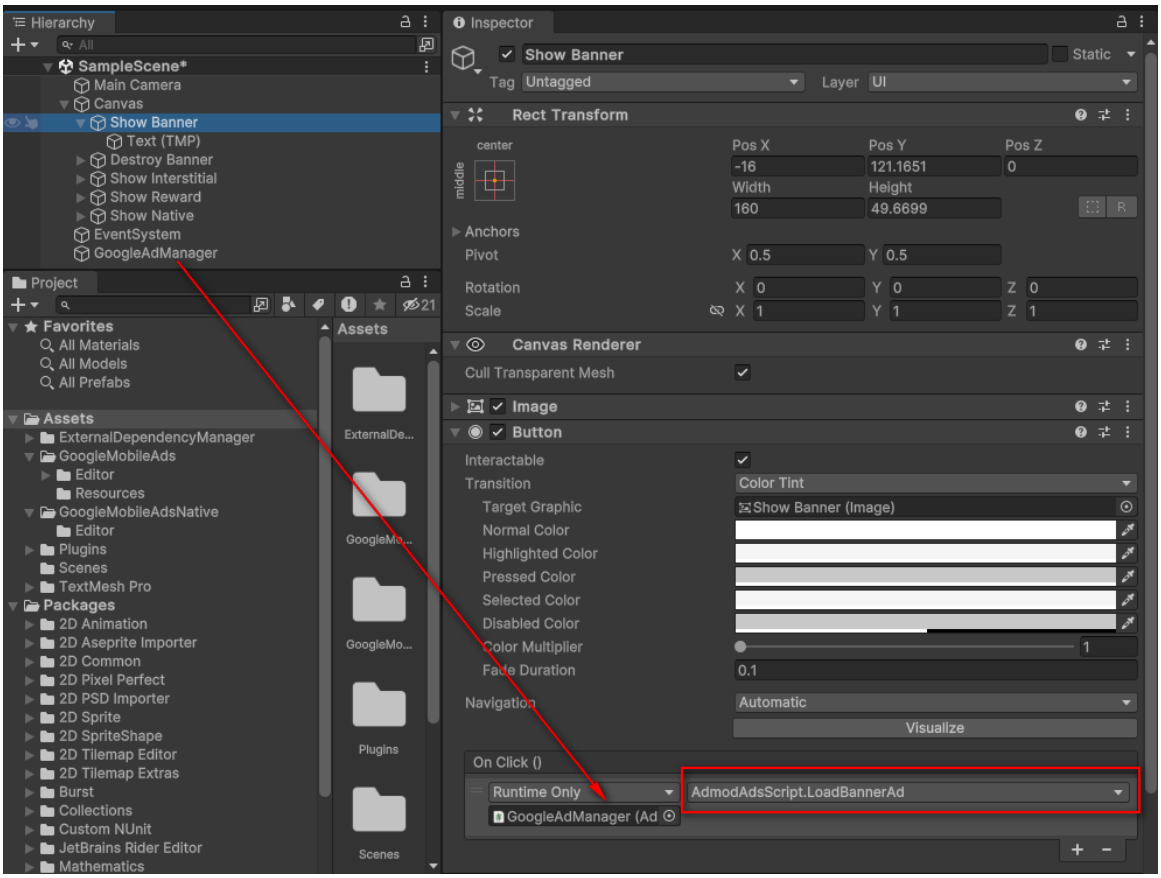
```
#region Rewarded
public void LoadRewardedAd()
{
    // Khởi tạo banner
    if (rewardedAd != null)
            DestroyRewardedAd();
    var adRequest = new AdRequest();
    adRequest.Keywords.Add("unity-admod-sample");
    RewardedAd.Load(rewardedId, adRequest, (RewardedAd ad, LoadAdError error) =>
    {
        if (error != null || ad == null) return;
        rewardedAd = ad;
        // lắng nghe các sự kiện
        RegisterEventHandlers(rewardedAd);
    });
    if (rewardedAd != null && rewardedAd.CanShowAd())
    {
        rewardedAd.Show((Reward reward) => { Debug.Log("Reward show");  });
    }
}
public void DestroyRewardedAd()
{
    rewardedAd.Destroy();
    rewardedAd = null;
}
private void RegisterEventHandlers(RewardedAd ad)...
#endregion
```

Hình 8.18. Mã nguồn để tải phần RewardedAd

Lưu ý, hàm RegisterEventHandlers() cũng có thể được copy từ trang web của Google Admob, phần Rewarded, Get started tại mục Listen to rewarded ad events hoặc để trống.

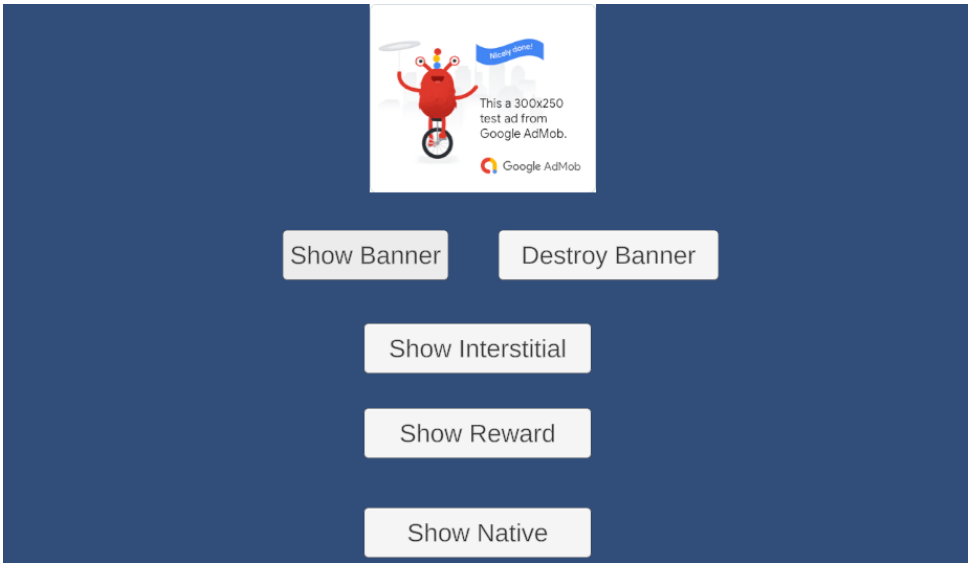
Riêng phần Native phức tạp hơn, không thuộc phạm vi cuốn sách này nên độc giả tìm hiểu thêm.

Bước 7: Chuyển qua Unity, vào trong từng nút, kéo đối tượng GoogleAdManager vào trong phần Object của sự kiện On Click(), lần lượt gọi các hàm tương ứng với từng nút như đã viết.



Hình 8.19. Gắn mã nguồn cho các nút

Chạy Game, bấm vào từng nút để xem kết quả như Hình 8.20

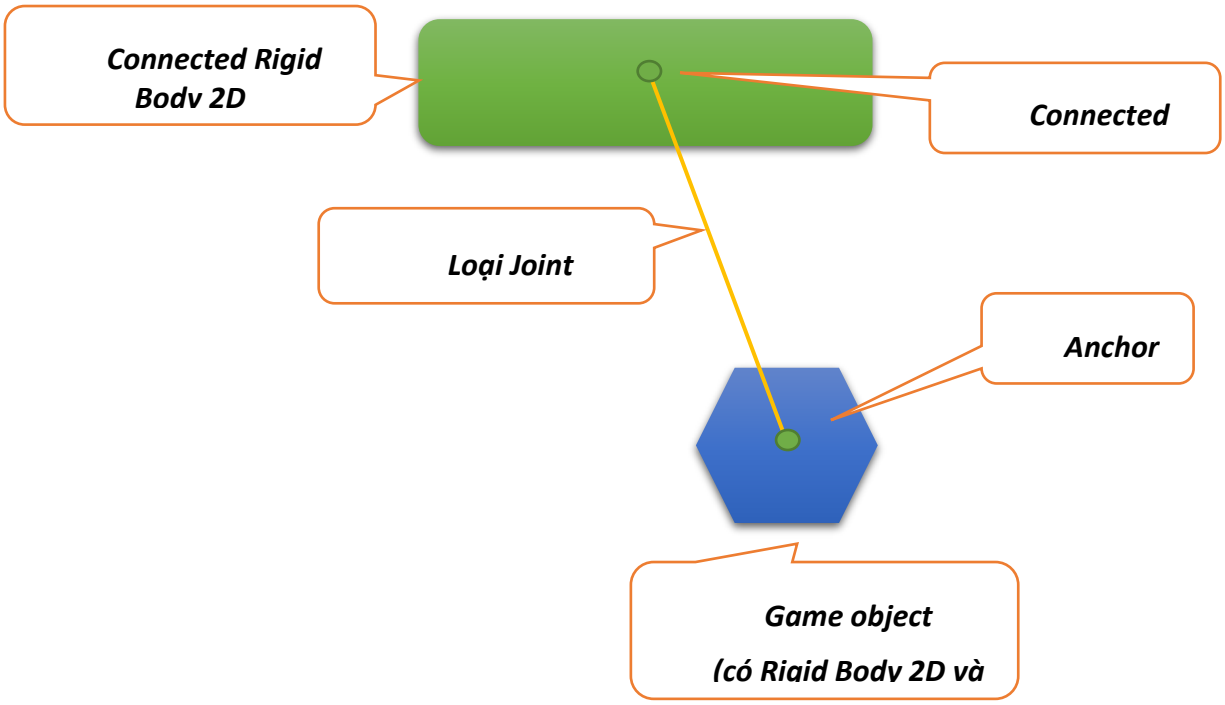


Hình 8.20. Chạy game và bấm Show Banner

4. Kết chương

TÀI LIỆU THAM KHẢO

https://docs.unity3d.com/


Connected Rigid Bodv 2D
Connected
Loại Joint
Anchor
Game object
(có Riaid Bodv 2D và