BỘ CÔNG THƯƠNG

TRƯỜNG ĐẠI HỌC KINH TẾ KĨ THUẬT CÔNG NGHIỆP

KHÓA LUẬN TỐT NGHIỆP

TÊN ĐỀ TÀI KHÓA LUẬN:

# PHÁT TRIỂN GAME 2D

**Ngành đào tạo: CÔNG NGHỆ THÔNG TIN Mã số ngành: 7480201**

**Họ và tên sinh viên: Nguyễn Đức Thịnh**

Người hướng dẫn khoá luận tốt nghiệp

# ThS. Vũ Thị Thu Huyền

**Hà Nội năm 2021**

# LỜI CAM ĐOAN

Em xin cam đoan:

1. Những nội dung trong luận văn là công trình nghiên cứu của em dưới sự hướng dẫn của giảng viên Vũ Thị Thu Huyền.
2. Mọi tham khảo trong luận văn đều được trích dẫn tên tác giả, tên công trình, thời gian, địa điểm công bố.
3. Các số liệu, kết quả trong luận văn là trung thực và chưa được ai công bố trong bất kỳ công trình nào khác.
4. Mọi sao chép không hợp lệ, vi phạm quy định đào tạo gian trá em xin chịu hoàn toàn trách nhiệm.

# LỜI CẢM ƠN

Em xin chân thành cảm ơn cô Vũ Thị Thu Huyền. Trong suốt quá trình làm luận văn cô đã hướng dẫn và chỉ bảo em tận tình. Cô đã giúp em chỉ ra những lỗ hổng và thiếu sót của bài luận. Em xin cảm ơn cô rất nhiều.

Đồng thời, em cũng chân thành cảm ơn các thầy cô khoa Công nghệ thông tin của trường Đại học Kinh tế - Kỹ thuật Công nghiệp đã dạy dỗ em những kiến thức chuyên môn trong 4 năm vừa qua để em có những kiến thức cơ sở và có nền tảng vững chắc nhằm hoàn thành bài luận.

Một lần nữa em xin chân thành cảm ơn.

*Hà Nội, ngày ....... tháng 2021*

# Sinh viên làm khóa luận

***Nguyễn Đức Thịnh***

# MỤC LỤC

[LỜI CAM ĐOAN i](#_TOC_250047)

[LỜI CẢM ƠN ii](#_TOC_250046)

[MỤC LỤC iii](#_TOC_250045)

[LỜI MỞ ĐẦU 1](#_TOC_250044)

* 1. LÝ DO CHỌN ĐỀ TÀI 1
  2. MỤC TIÊU NGHIÊN CỨU 1
  3. ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU CỦA ĐỀ TÀI 1
  4. PHƯƠNG PHÁP NGHIÊN CỨU 1
  5. PHẠM VI NGHIÊN CỨU. 1
  6. CẤU TRÚC LUẬN VĂN 2

CHƯƠNG 1 : TỔNG QUAN VỀ GAME ENGINE VÀ ENGINE UNITY3

* 1. TỔNG QUAN VỀ GAME ENGINE 3
     1. [Khái niệm 3](#_TOC_250043)
     2. [Mục đích 3](#_TOC_250042)
     3. [Lịch sử 4](#_TOC_250041)
     4. [Xu hướng hiện tại 5](#_TOC_250040)
     5. [Phần mềm trung gian 5](#_TOC_250039)
  2. TỔNG QUAN VỀ ENGINE UNITY 6
     1. [Unity là gì ? 6](#_TOC_250038)
     2. [Sơ lược lịch sử hình thành và phát triển của Unity 8](#_TOC_250037)
        1. Hình thành 8
        2. Giải Thưởng 9
        3. Khách Hàng 9
     3. [Các khái niệmtrong Unity 10](#_TOC_250036)
        1. Assets 10
        2. Scenes 10
        3. Game Object 10
        4. Components 10
        5. Script 11
        6. Prefabs 11
     4. [Tính năng của engine Unity 11](#_TOC_250035)
  3. NGÔN NGỮ TRONG UNITY 12
     1. [Ngôn ngữ C# 12](#_TOC_250034)
        1. Tổng quan về ngôn ngữ C# 12
        2. Đặc điểm ngôn ngữ 13
        3. Lịch sử 13
     2. [Ngôn ngữ Javascript 14](#_TOC_250033)
        1. Tổng quan về ngôn ngữ Javascript 14
        2. Ứng dụng 14

[CHƯƠNG 2 LẬP TRÌNH VỚI UNITY 16](#_TOC_250032)

* 1. GIAO DIỆN UNITY 16
     1. [Khởi tạo project 16](#_TOC_250031)
     2. [Giao diện chính 16](#_TOC_250030)
     3. [Thanh công cụ 17](#_TOC_250029)
     4. [Tab Scene 18](#_TOC_250028)
     5. [Gizmo 18](#_TOC_250027)
     6. [Tab Game 18](#_TOC_250026)
     7. [Tab Project 19](#_TOC_250025)
     8. [Tab Inspector 20](#_TOC_250024)
     9. [Tab Hierarchy 21](#_TOC_250023)
  2. GAME OBJECT 21
     1. [Các thuộc tính và hàm 21](#_TOC_250022)
        1. Transform 21
        2. tag-name 21
        3. Layer 23
        4. Active 23
        5. isStatic 23
        6. Tìm kiếm GameObject 24
        7. Khởi tạo Game Object 25
        8. Giải phóng Game Object 27
     2. [Thao tác với Component 28](#_TOC_250021)
        1. GetComponent và GetComponents 28
        2. Tìm kiếm Component trong các đối tượng cha hoặc con 29
  3. CƠ CHẾ VẬT LÝ TRONG UNITY 29
     1. [Rigidbody 30](#_TOC_250020)
     2. [Colliders 31](#_TOC_250019)
  4. LỚP INPUT 33
     1. [Sự kiện OnMouse 33](#_TOC_250018)
     2. [Sự kiện GetMouseButton 34](#_TOC_250017)
     3. [Nhập dữ liệu từ bàn phím 34](#_TOC_250016)
        1. GetKey và GetButton 35
        2. GetAxis 35
  5. CÁC HỆ TỌA ĐỘ TRONG UNITY 36
     1. [World Coordinate 36](#_TOC_250015)
     2. [Viewport Coordinate 36](#_TOC_250014)
     3. [Screen Coordinate 37](#_TOC_250013)
     4. [UI Coordinate 37](#_TOC_250012)
     5. [Chuyển đổi giữa các hệ toạ độ trong Unity 37](#_TOC_250011)
  6. CAMERA 38
     1. [Projection 38](#_TOC_250010)
     2. [Field of view (fov) 38](#_TOC_250009)
     3. [Tùy chỉnh Camera tại tab Inspector 39](#_TOC_250008)
  7. ÂM THANH TRONG UNITY 40
     1. [Audio Source Component 40](#_TOC_250007)
     2. [Một số hàm thông dụng 43](#_TOC_250006)
  8. COROUTINE 44
     1. [Từ khoá yield 44](#_TOC_250005)
     2. [Phương thức WaitForSeconds 45](#_TOC_250004)
     3. [Sử dụng Coroutine trong Unity 45](#_TOC_250003)
  9. CÁC HÀM SỰ KIỆN VÀ THỨ TỰ HOẠT ĐỘNG 46

[CHƯƠNG 3 : XÂY DỰNG GAME VỚI UNITY 50](#_TOC_250002)

* 1. Ý TƯỞNG 50
  2. MÔ TẢ TRÒ CHƠI 50
  3. XÂY DỰNG CHƯƠNG TRÌNH 55
  4. NHẬN XÉT 59

[PHẦN KẾT LUẬN 60](#_TOC_250001)

[TÀI LIỆU THAM KHẢO 62](#_TOC_250000)

**DANH MỤC CÁC HÌNH**

Hình 2.1 Khởi tạo project (bản 2018.3.6f1) 16

Hình 2.2 Giao diện chính của Unity 16

Hình 2.3 Tab Scene 18

Hình 2.4 Tab game 18

Hình 2.5 Tab Project 19

Hình 2.6 Tab Inspector 20

Hình 2.7 Tab Hierachy 21

Hình 2.8 Danh sách các tag 22

Hình 2.9 Thêm tag 22

Hình 2.10 Layer 23

Hình 2.11 Môi trường được tích Static 24

Hình 2.12 Danh sách các PrimitiveType được hỗ trợ 26

Hình 2.13 Dùng hàm Instantiate để tạo ra các Zombie đã được đặt các thuộc tính sẵn 27

Hình 2.14 Rigidbody 30

Hình 2.15 Các loại Colliders 31

Hình 2.16 Cài đặt Input trong Unity 35

Hình 2.17 Tạo đối tượng Camera 38

Hình 2.18 Minh họa Field of view (fov) 39

Hình 2.19 Tùy chỉnh camera trong Unity 39

Hình 2.20 Tùy chỉnh Audio trong Unity 41

Hình 2.21 Danh mục các định dạng âm thanh được hỗ trợ 43

Hình 2.22 Các hàm sự kiện và thứ tự hoạt động 47

Hình 3.1 Các Zombie Bình thường. 50

Hình 3.2 Các Zombie chạy nhanh 51

Hình 3.3 Các Zombie nhiều máu 51

Hình 3.4 Bản đồ nhỏ 52

Hình 3.5 Các đồ vật trợ giúp cho người chơi , từ trái qua phải là Hộp máu, Hộp đạn, Lựu đạn. 53

Hình 3.6 Trạng thái của người chơi khi nhặt đồ vật trợ giúp 53

Hình 3.7 Người chơi bắn hạ Zombie và rơi ra đồ 53

Hình 3.8 Các loại súng khác nhau của người chơi 54

Hình 3.9 Súng tầm xa. 54

Hình 3.10 Súng cự li gần. 55

Hình 3.11 Súng phóng lựu. 55

Hình 3.12 Người chơi mất máu khi Zombie chạm vào 55

Hình 3.13 Điểm số và mục tiêu của màn chơi 55

Hình 3.14 Scene Menu chính 56

Hình 3.15 Hướng dẫn cách chơi 56

Hình 3.16 Chọn màn chơi để vào game. 57

Hình 3.17 Một trong các màn chơi trong game. 57

Hình 3.18 Tạm dừng game khi đang chơi 58

Hình 3.19 Khi người chơi thua. 58

Hình 3.20 Khi người chơi thắng. 59

# LỜI MỞ ĐẦU

1. LÝ DO CHỌN ĐỀ TÀI

Trong thời đại công nghệ thông tin như hiện nay, sản phẩm công nghệ ngày càng chịu sự đánh giá khắt khe hơn từ phía những người dùng, đặc biệt là về sản phẩm Game vốn nhận được rất nhiều sự đánh giá từ phía các Game thủ, và cả những người chơi bình thường. Ngành công nghiệp Game hiện nay có thể nói là bùng nổ, với tốc độ phát triển đến chóng mặt, rất nhiều những Game hay và hấp dẫn đã được ra đời trong thời gian qua. Phía sau những Game phát triển và nổi tiếng như vậy đều có một Game Engine. Game Engine là một công cụ hỗ trợ, một Middleware giúp người phát triển viết Game một cách nhanh chóng và đơn giản, đồng thời cung cấp khả năng tái sử dụng các tài nguyên và mã nguồn cao do có thể phát triển nhiều Game từ một Game Engine.

Từ xu hướng phát triển ở trên, luận văn sẽ khảo sát và nghiên cứu về Unity – một Game Engine rất phổ biến và mạnh mẽ hiện nay đang được nhiều nhà phát triển sử dụng.Đây cũng là Game Engine em sử dụng để phát triển game Alien Shooter trong quá trình nghiên cứu.

1. MỤC TIÊU NGHIÊN CỨU
   * Nghiên cứu về các thành phần, cấu trúc, phương thức hoạt động của Unity như các đối tượng 2D, 3D, quản lý dữ liệu đầu vào, lưu trữ, âm thanh… để ứng dụng vào lập trình game.
   * Xây dựng ứng dụng game Alien Shooter hoàn chỉnh dựa trên kết quả tìm hiểu được.
2. ĐỐI TƯỢNG VÀ PHẠM VI NGHIÊN CỨU CỦA ĐỀ TÀI
   * Cơ sở kiến thức, lý thuyết về Unity.
   * Phát triển game đa nền tảng sử dụng Unity.
3. PHƯƠNG PHÁP NGHIÊN CỨU
   * Phương pháp nghiên cứu tài liệu.
   * Phương pháp thực hành.
4. PHẠM VI NGHIÊN CỨU.
   * Nghiên cứu lý thuyết về Unity.
   * Phát triển game 2D dựa trên Unity Engine cho nền tảng PC/Android.
5. CẤU TRÚC LUẬN VĂN.

Ngoài phần mở đầu và phần kết luận, luận văn gồm 3 chương:

* + Chương 1: Tổng quan về Game Engine và Unity Engine: Phần này giới thiệu một cách tổng quan về Game Engine và Unity.
  + Chương 2: Lập trình với Unity: Phần này trình bày về các thành phần và phương thức hoạt động của Unity như : giao diện , các đối tượng 2D, đối tượng Game , dữ liệu vào, âm thanh, lưu trữ….
  + Chương 3: Xây dựng game với Unity : Phần này trình bày về quá trình xây dựng game bao gồm: ý tưởng, các đối tượng đồ họa, âm thanh sử dụng, game logic, lập trình, …
  + Phần kết luận: Phần này đánh giá ưu, nhược điểm và hướng phát triển tiếp của đề tài.

# Chương 1 : TỔNG QUAN VỀ GAME ENGINE VÀ ENGINE UNITY

* 1. TỔNG QUAN VỀ GAME ENGINE

# Khái niệm

Một Game Engine (hay công cụ tạo Game / động cơ Game) là một phần mềm được viết cho mục đích thiết kế và phát triển video Game. Có rất nhiều loại Game Engine dùng để thiết kế Game cho các hệ máy như hệ Consoles hay máy tính cá nhân (PC) hay trên điện thoại (Mobile). Chức năng cốt lõi của Game Engine phần lớn nằm trong công cụ dựng hình (kết xuất đồ họa) cho các hình ảnh 2 chiều (2D) hay 3 chiều (3D), công cụ vật lý (hay công cụ tính toán và phát hiện va chạm), âm thanh, mã nguồn, hình ảnh động (Animation), trí tuệ nhân tạo, phân luồng, tạo dòng dữ liệu xử lý, quản lý bộ nhớ, dựng ảnh đồ thị và kết nối mạng. Quá trình phát triển Game tiết kiệm được rất nhiều thời gian và kinh phí vào việc tái sử dụng và tái thích ứng một Engine để tạo nhiều Game khác nhau.

# Mục đích

Game Engine cung cấp một bộ các công cụ phát triển trực quan và có thể tái sử dụng từng thành phần trong đó. Nói chung các bộ công cụ này cung cấp một môi trường phát triển tích hợp được đơn giản hóa. Phát triển ứng dụng nhanh (Rapid Application Development) cho Game theo cách lập trình hướng dữ liệu. Những Game Engine này đôi khi còn được gọi là các "phần mềm trung gian cho Game" (Game Middleware), như ý nghĩa của thuật ngữ, chúng cung cấp một nền tảng phần mềm linh hoạt và dễ dàng sử dụng lại với mọi chức năng cốt lõi cần thiết ngay trong nó để có thể phát triển một ứng dụng Game đồng thời giảm giá thành, độ phức tạp, và kịp thời hạn phát hành - tất cả các yếu tố quan trọng trong ngành công nghiệp Game đầy cạnh tranh. Giống như các phần mềm trung gian khác, Game Engine thường cung cấp một nền tảng trừu tượng hóa, cho phép một Game có thể chạy trên nhiều hệ máy bao gồm các hệ console hoặc máy tính cá nhân với một vài thay đổi (nếu cần) trong mã nguồn của Game đó. Thông thường, phần mềm trung gian cho Game được thiết kế với một nền tảng kiến trúc dựa trên các thành phần khác, cho phép các hệ thống khác nhau trong Engine có thể thay thế hoặc mở rộng với các phần mềm trung gian khác chuyên biệt hơn như là Havok cho hệ thống vật lý trong Game, Miles Sound System cho âm thanh, hay Bink cho các đoạn video. Một số Game Engine chỉ cung cấp khả năng dựng hình (kết xuất) 3D thời gian thực hay một khả năng riêng biệt nào khác thay vì rất nhiều chức năng trong phạm vi rộng mà Game yêu cầu. Loại Engine này thường được gọi là: "Graphics Engine", "Rendering Engine," hay "3D Engine" thay vì thuật ngữ bao quát hơn là "Game Engine”. Một vài ví dụ cho các Engine đồ họa là: RealmForge, Truevision3D, OGRE, Crystal Space, Genesis3D, Vision Engine, Irrlicht và JMonkey Engine.

Thông thường, Engine 3D hay hệ thống kết xuất đồ họa trong Game Engine thường được xây dựng trên một nền tảng giao diện lập trình ứng dụng đồ họa như Direct3D hay OpenGL cung cấp sẵn một hệ ảo hóa cho GPU hay card đồ họa. Thư viện mã nguồn cấp thấp (cấp cơ bản) như DirectX, SDL, và OpenAL (thư viện âm thanh mã nguồn mở) cũng thường được sử dụng trong các Game khi chúng cung cấp khả năng truy cập phần cứng độc lập.

# Lịch sử

Thuật ngữ "Game Engine" xuất hiện vào giữa những năm 90, đặc biệt là trong mối quan hệ giữa Game 3D. Như các thuơng hiệu nổi tiếng của id Sortware: Doom và Quake, thay vì phải làm việc từ đầu, các nhà phát triển khác (nếu được cấp phép) sẽ có quyền truy nhập vào phần lõi (mã nguồn) của Game và thiết kế các hình ảnh, nhân vật, vũ khí, và các màn chơi của riêng họ

- gọi là Game Content (nội dung Game) hay "Game Assets" (tài sản Game). Các Game sau này, như Quake III Arena và sản phẩm năm 1998 của Epic Games: Unreal được thiết kế với cách tiếp cận mới này, Game Engine và nội dung Game được chia thành các phần riêng biệt để phát triển. Engine có thể tái sử dụng khiến việc phát triển Game tiếp theo nhanh hơn và dễ dàng hơn, một thuận lợi to lớn trong ngành công nghiệp cạnh tranh này. Game Engine hiện đại là một trong những ứng dụng được viết ra (bằng các ngôn ngữ lập trình) phức tạp nhất, thường xuyên phải có rất nhiều tinh chỉnh trong hệ thống để đảm bảo kiểm soát chính xác trải nghiệm người dùng. Sự phát triển liên tục của Game Engine đã tạo ra một sự phân chia mạnh mẽ giữa các công việc dựng hình, viết kịch bản, thiết kế hình ảnh, và thiết kế màn chơi. Hiện nay thông thường một đội ngũ phát triển Game điển hình phải có số lượng họa sĩ gấp vài lần số lượng lập trình viên.

# Xu hướng hiện tại

Với công nghệ tạo Game Engine càng phát triển và trở nên thân thiện hơn với người sử dụng, ứng dụng của nó càng được mở rộng, và giờ đây được sử dụng để tạo các Game mang mục đích khác với giải trí đơn thuần như: mô phỏng, huấn luyện ảo, y tế ảo, và mô phỏng các ứng dụng quân sự. Nhiều Game Engine đang được tạo ra bằng các ngôn ngữ lập trình cấp cao như Java và C# hay .NET (ví dụ: TorqueX, và Visual3D.NET) hay Python (Panda3D). Vì hầu hết các Game 3D hiện nay đều có giới hạn cho GPU (giới hạn bởi sức mạnh của card đồ họa), khả năng gây chậm máy của các ngôn ngữ lập trình cấp cao trở nên không đáng kể, trong khi việc tăng năng suất được cung cấp bởi các ngôn ngữ này lại có lợi cho các nhà phát triển Game Engine.

# Phần mềm trung gian

Một số hãng hiện nay chuyên tập trung vào phát triển các bộ phần mềm gọi là phần mềm trung gian (Middleware). Các nhà phát triển Middleware cố gắng phát triển một bộ phần mềm mạnh mẽ bao gồm hầu hết các yếu tố mà một nhà phát triển Game cần đến để xây dựng một Game. Phần lớn các Middleware cung cấp các cơ sở để phát triển game một cách dễ dàng, như hình ảnh, âm thanh, hệ thống vật lý và chức năng AI. Gamebryo và RenderWare là hai phần mềm trung gian được sử dụng khá rộng rãi. Một số Middleware chỉ làm một chức năng nhưng nó còn hiệu quả hơn cả toàn bộ Engine. Ví dụ, SpeedTree được sử dụng để dựng hình ảnh cây cối thêm thực tế trong Game nhập vai The Elder Scrolls IV: Oblivion. Bốn gói phần mềm trung gian được sử dụng rộng rãi nhất, thường được cung cấp dưới dạng một chức năng bổ sung trong một hệ thống Engine gồm có Bink của hang RAD Game Tools, hãng Firelight với FMOD, Havok, và GFx của Scaleform. RAD Game Tools phát triển Bink cho dựng hình video cơ bản, đi cùng với âm thanh Miles, và dựng hình 3D Granny. FMOD của Firelight là một thư viện và bộ công cụ phần mềm âm thanh mạnh mẽ giá rẻ. Havok cung cấp một hệ thống mô phỏng vật lý mạnh, cùng với các bộ phần mềm về hình ảnh động và mô phỏng hành vi. Scaleform đem lại GFx dùng cho các biểu diễn cao cấp trên nền giao diện Flash, cùng với các giải pháp trình chiếu video chất lượng cao, và một Input Method Editor (IME) phần bổ sung nhằm hỗ trợ trò chuyện trong Game bằng ngôn ngữ Châu Á.

* 1. TỔNG QUAN VỀ ENGINE UNITY

# Unity là gì ?

Unity là một phần mềm làm game đa nền tảng được phát triển bởi Unity Technologies, mà chủ yếu để phát triển video game cho máy tính, consoles và điện thoại. Lần đầu tiên nó được công bố chạy trên hệ điều hành OS X, tại Apple's Worldwide Developers Conference vào năm 2005, đến nay đã mở rộng 27 nền tảng.

6 phiên bản chính của phần mềm này đã được phát hành. Tại triển lãm WWDC năm 2006, Apple đã trao thưởng giải Best Use of Mac OS X Graphics cho ứng dụng này.

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

Unity hỗ trợ đồ họa 2D và 3D, các chức năng được viết chủ yếu qua ngôn ngữ C#. Hai ngôn ngữ lập trình khác cũng được hỗ trợ: Boo, đã bị loại cùng với việc phát triển Unity 5 and UnityScript bị loại vào tháng 8 năm 2017 sau khi phát hành Unity 2017.1. UnityScript là một ngôn ngữ lập trình độc quyền có cú pháp tương tự JavaScript. Phần mềm nhắm mục tiêu các đồ họa APIs sau: Direct3D trên Windows và Xbox One; OpenGL trên Linux, macOS, và Windows; OpenGL ES trên Android và iOS; WebGL trên web; và APIs độc quyền trên các máy chơi video game. Ngoài ra, Unity hỗ trợ APIs cấp thấp như Metal trên iOS và macOS và Vulkan trên Android, Linux, và Windows, cũng như Direct3D 12 trên Windows và Xbox One. Trong 2D games, Unity cho phép nhập sprites và một renderer thế giới 2D tiên tiến. Đối với 3D games, Unity cho phép thiết lập các đặc điểm kĩ thuật của các kết cấu và độ phân giải mà công cụ trò chơi hỗ trợ, cung cấp các hỗ trợ cho bump mapping, reflection mapping, parallax mapping, cảnh không gian ambient occlusion (SSAO), hiệu ứng bóng đổ bằng cách sử dụng shadow maps, render thiết lập toàn cảnh đến hiệu ứng. Unity cũng cung cấp các dịch vụ cho nhà phát triển, bao gồm: Unity Ads, Unity Analytics, Unity Certification, Unity Cloud Build, Unity Everyplay, Unity IAP, Unity Multiplayer, Unity Performance Reporting and Unity Collaborate.

Unity nổi bật với khả năng xây dựng trò chơi chạy trên nhiều nền tảng. Các nền tảng được hỗ trợ hiện nay là Android, Android TV, Facebook Gameroom, Fire OS, Gear VR, Google Cardboard, Google Daydream, HTC Vive, iOS, Linux, macOS, Microsoft HoloLens, Nintendo 3DS family , Nintendo Switch, Oculus Rift, PlayStation 4, PlayStation Vita, PlayStation VR, Samsung Smart TV, Tizen, tvOS, WebGL, Wii U, Windows, Windows Phone, Windows Store, và Xbox One. Unity trước đây cũng đã hỗ trợ 7 nền tảng khác chạy trên Unity Web Player. Unity Web Player là một plugin của trình duyệt chạy trên Windows và OS X, đã bị gỡ bỏ vì lợi ích của WebGL.

Unity là bộ công cụ phát triển phần mềm mặc định (SDK) cho máy chơi game video game Wii U của Nintendo, kèm theo bản miễn phí của Nintendo với mỗi giấy phép phát triển Wii U. Unity Technologies gọi việc kết hợp sản xuất SDK với một bên thứ ba là "industry first".

Bỏ ra khoản tiền 1800USD là phiên bản pro đã nằm trong tay của chúng ta, dĩ nhiên tại Việt Nam số tiền này vẫn là quá lớn nhưng thật may là đã có phiên bản Unity Free. Tuy nhiên, nhiều tính năng quan trọng (Network) bị cắt giảm nhưng đó không phải là vấn đề quá lớn nếu muốn phát triển một tựa game tầm trung.

Vào năm 2009, Unity nằm trong top 5 game engine tốt nhất cho việc sản xuất game với chỉ sau 4 năm phát triển. Unity đứng thứ 4, xếp sau Unreal Engine 3, Gamebryo Engine (được VTC Studio mua về phát triển SQUAD) và Cry Engine 2. Lượng tài liệu hướng dẫn Unity rất phong phú. Hơn thế nữa nó còn có sẵn một cộng đồng cực lớn với diễn đàn riêng. Bất cứ điều gì không hiểu chúng ta đều có thể thoải mái hỏi và nhận được câu trả lời nhanh chóng, tận tâm.

Quá trình tạo địa hình cũng như truy xuất từ các phần mềm 3DSMax, Maya, Cinema4D... rất nhanh chóng. Sức mạnh và sự tiện lợi của Unity là vô cùng lớn.

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

Sự tiện lợi: Những chuyên dùng 3Dmax, hay Maya hoặc phần mềm mã nguồn mở Blender thì quả là thật tuyệt, chúng ta sẽ có một lợi thế lớn khi viết game trên Unity này, bởi công việc tạo các mô hình 2D, 3D sẽ trở lên dễ dàng hơn rất nhiều, việc kết hợp giữa người lập trình và người thiết kế các mô hình sẽ nhanh và hiệu quả hơn. Trong Unity chúng ta có thể import trực tiếp các file mô hình đang thiết kế và sẽ thiết kế hoàn thiện tiếp nếu chưa xong trong khi đó công việc import chỉ diễn ra một lần. Không như việc phải dùng các công cụ khác để thực hiện viết game chúng ta sẽ phải xuất chúng ra một dạng nào đó và mỗi lần sửa lại phần mô hình chúng ta lại phải import lại, và như thế là quá mất thời gian trong việc tạo và chỉnh sửa các mô hình theo ý muốn. Ngoài ra Unity còn cho chúng ta trực tiếp tạo các mô hình nếu muốn. Việc đặt các thuộc tính vật lý trong Unity cũng cực kỳ dễ dàng và hỗ trợ sẵn nhiều chức năng.

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

* + - 1. *Hình thành*

Phần lõi của Unity ban đầu được viết bởi Joachim Ante vào năm 2001. Sau đó công ty được hình thành vào năm 2005 và bắt đầu với phiên bản 1.0. Đến năm 2007, Unity được nâng lên phiên bản 2.0. Unity bắt đầu hỗ trợ iPhone vào năm 2008. Vào tháng 6/2010, Unity chính thức hỗ trợ Android và cho ra đời phiên bản 3.0 có hỗ trợ Android vào tháng 9/2010 và bây giờ là phiên bản Unity 5. Có thể thấy tốc độ phát triển của Unity khá nhanh.

Năm 2012, VentureBeat phát biểu rằng "rất ít công ty đóng góp vào sự phát triển trò chơi độc lập như Unity Technologies.... hơn 1.3 triệu người dùng đang sử dụng Unity để phát triển game trên iOS, Android, máy chơi game, PC, và trò chơi trên web.... Unity sẽ mở ra thời kỳ phát triển công cụ phát triển trò chơi đa nền tảng."

Đối với Apple Design Awards tại cuộc triển lãm thương mại WWDC 2006, Apple, Inc. đã gọi Unity là Best Use of Mac OS X Graphics (tạm dịch: Ứng dụng tốt nhất cho đồ họa trên nền tảng Mac OS X), một năm sau khi Unity ra mắt tại cùng một triển lãm thương mại Unity Technologies cho biết đây là lần đầu tiên một công cụ phát triển game được trao giải thưởng này.Một cuộc khảo sát tháng 5 năm 2012 của tạp chí Game Developer đã khẳng định Unity là công cụ phát triển trò chơi hàng đầu trên di động.

Unity 5 cũng nhận được những lời khen tương tự, với The Verge đã nói rằng "Unity giúp việc phát triển game trở nên dễ dàng và phổ biến Unity 5

là một bước tiến đáng mong đợi trong tương lai."

Sau khi phát hành Unity 5, Unity Technologies bị chỉ trích rằng khổi lượng lớn các game phát hành trên Steam là của các nhà phát triển thiếu kinh nghiệm. CEO John Riccitiello giải thích trong một cuộc họp báo rằng đây là tác dụng phụ khi chúng ta xã hội hóa việc phát triển game: "If I had my way, I'd like to see 50 million people using Unity – although I don't think we're going to get there any time soon. I'd like to see high school and college kids using it, people outside the core industry. I think it's sad that most people are consumers of technology and not creators. The world's a better place when people know how to create, not just consume, and that's what we're trying to promote." (tạm dịch: nếu chúng tôi có cách, tôi muốn 50 triệu người sử dụng Unity- dù chúng tôi biết rất khó đạt được điều đó. Tôi muốn thấy những đứa trẻ ở trường Trung học và Đại học sử dụng nó, những người ngoài ngành. Tôi nghĩ rằng thật buồn khi hầu hết mọi người là người tiêu dùng về công nghệ, chứ không phải người sáng tạo. Thế giới là một nơi tốt đẹp hơn khi người ta biết làm thế nào để tạo ra, không chỉ tiêu thụ, và đó là những gì chúng tôi đang cố gắng để thúc đẩy.)

Vào tháng 12 năm 2016, Unity Technologies thông báo họ sẽ thay đổi hệ thống số phiên bản trên Unity từ định theo thứ tự sang năm phát hành để sắp xếp các phiên bản với nhịp cập nhật thường xuyên hơn

* + - 1. *Giải Thưởng*

Unity đã đoạt được nhiều giải lớn với những giải chính sau:

Năm 2006, Unity đạt "Best Use of Mac OS X Graphics" tại Apple's WWDC. Đây là lần đầu tiên một công cụ phát triển game đạt được chất lượng do giải thưởng uy tín này đưa ra.

Tháng 7/2014, Unity đã đạt giải "Best Engine" tại liên hoan trao giải thưởng công nghệ hàng năm của Anh.

Năm 2009, Unity Technologies có tên trong "Top 5 công ty game của năm" do Gamasutra tổ chức.

Năm 2010, Unity đoạt giải Best Engine Finalist do Develop Magazine bình chọn, giải Technology Innovation Award của Wall Street Journal ở thể loại phần mềm.

* + - 1. *Khách Hàng*

Unity được trên 250.000 người đăng ký sử dụng gồm Bigpoint, Cartoon Network, Coca-Cola, Disney, Electronic Arts, LEGO, Microsoft, NASA, Ubisoft, Warner Bros, các hãng phim lớn nhỏ, các chuyên gia độc lập, sinh viên và những người đam mê.

# Các khái niệmtrong Unity

* + - 1. *Assets*

Assets là những tài nguyên được sử dụng trong việc phát triển game trong Unity.

Các Assets này có thể là hình ảnh, mô hình 3D, âm thanh, hiệu ứng, ... được tạo ra bởi các nhà phát triển, có thể được download miễn phí hoặc trả phí. Tính năng này giúp giảm thiểu rất nhiều thời gian cho việc thiết kế và lập trình game. Từ những tập tin hình ảnh, mô hình 3D đến các tập tin âm thanh. Unity gọi các tập tin mà chúng ta dùng để tạo nên trò chơi là tài sản (Asstets). Điều này lý giải tại sao tất cả các tập tin, thư mục của các dự án Unity đều được lưu trữ trong một thư mục có tên “Assets”.

* + - 1. *Scenes*

Trong Unity, Scene là một cảnh game, không gian game nơi thiết lập bố cục của các GameObjects, hoặc là một phần chứa các thiết lập giao diện như các menu trong game.

Tạo ra nhiều Scenes sẽ giúp linh hoạt trong việc phân phối tối ưu tài nguyên, quản lý các phân đoạn trong game một cách độc lập.

* + - 1. *Game Object*

Unity định nghĩa GameObject là đối tượng đại diện cho các Assets trong game như nhân vật, cây cỏ, công cụ, đạo cụ, camera, hiệu ứng ... Các GameObject đều chứa một thuộc tính cơ bản là Transform, dùng để thiết lập vị trí, hướng và kích thước của GameObject.

* + - 1. *Components*

Components là các thuộc tính thêm vào GameObject như là Animation, âm thanh, mô hình 3D, hiệu ứng, … nhằm xây dựng, kết hợp các yếu tố khác nhau để xác định các hình thái, hành vi, hiệu ứng, ... của đối tượng mong muốn trong game..

Bằng cách “gắn” (attach) chúng vào trong Game Object, chúng ta có thể ngay lập tức áp dụng tác động của chúng lên đối tượng. Những Components phổ biến trong quá trình phát triển trò chơi đều được Unity hỗ trợ sẵn. Ví dụ như thành phần Rigitbody chuyên xử lí các vấn đề vật lý, các yếu tố đơn giản đến từ thực tế khác như ánh sáng, camera, … Để tạo nên các yếu tố tương tác trong trò chơi, chúng ta sẽ sử dụng Script (mã kịch bản), chúng cũng được xem như là một Components trong Unity.

* + - 1. *Script*

Scripts là một Component trong Unity. Đây là thành phần thiết yếu dùng để tương tác với các hành động của người chơi, hoặc quản lý các sự kiện để thay đổi chiều hướng của game tương ứng với kịch bản game.

Unity cung cấp cho lập trình viên khả năng viết Script bằng ngôn ngữ C#.

Được Unity xem như một Component, Script là một thành phần thiết yếu trong quá trình phát triển trò chơi và đáng đề cập như một khái niệm “chìa khóa”. Unity cung cấp cho chúng ta khả năng viết Script bằng cả 3 ngôn ngữ là JavaScript, C# và Boo (một dẫn xuất của ngôn ngữ Python).

Unity không đòi hỏi chúng ta phải học làm thế nào để lập trình trong Unity, nhưng hầu như chúng ta phải sử dụng Script tại mỗi thành phần trong kịch bản mà chúng ta phát triển. Unity đã xây dựng sẵn một tập hợp đa dạng các Class, Function, … mà chúng ta hoàn toàn có thể ứng dụng trong quá trình lập trình trò chơi của mình.

Để viết Script, chúng ta sẽ làm việc với một trình biên tập Script độc lập của Unity, hoặc với chương trình Mono Developer được tích hợp và đồng bộ với Unity trong những phiên bản mới nhất gần đây.

* + - 1. *Prefabs*

Prefabs là một GameObject hoàn chỉnh sau khi thêm các thuộc tính và được lưu trữ lại để tái sử dụng. Các GameObject được nhân bản từ một prefab sẽ giống nhau hoàn toàn.

Là một thành phần khá quan trọng, cho phép chúng ta tạo dựng các gói tài nguyên mà có thể tận dụng lại ở mọi lúc mà chúng ta cần, Prefabs cho phép chúng ta lưu trữ các đối tượng với những Components và những thiết đặt hoàn chỉnh. Có thể so sánh với khái niệm MovieClip trong Adobe Flash, Prefabs chỉ đơn giản là một đối tượng chứa (Container) mà chúng ta có thể đưa bất kỳ một đối tượng hay dữ liệu mẫu nào mà chúng ta muốn tái sử dụng về sau.

* + - 1. *Camera*

Camera trong Unity dùng để thể hiện khung hình, góc nhìn mà người chơi có thể nhìn thấy được trong game.

Trong một game có thể thiết lập nhiều camera để chia nhỏ màn hình người chơi, tạo các hiệu ứng, hoặc tùy chỉnh như nhìn được từ phía sau, bản đồ thu nhỏ, …

* + - 1. Light

Light là một phần thiết yếu của mỗi scene. Light xác định màu sắc và độ sáng của môi trường Game. Việc kết hợp nhiều lights trong scene giúp tạo ra các hiệu ứng ánh sáng phù hợp với từng trò chơi.

# Tính năng của engine Unity

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

Triển khai được trên nhiều nền tảng:

Chương trình độc lập trên Windows và Mac OS.

Trên web, thông qua Unity Web Player plugin cho Internet Explorer, Firefox, Safari, Opera, Chrome, cho cả Windows và Mac OS.

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

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

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

Unity Asset Server: Đây là một tính năng khá mới của Unity, theo đó Unity sẽ cung cấp một hệ thống quản lý theo dạng phiên bản cho tất cả asset và cả script. Đây là một kho chứa các tài nguyên cần thiết cho việc làm game. Khi import cũng như sửa chữa, trạng thái của asset ngay lập tức được cập nhật. Server chạy trên database opensource PostgreSQL và có thể truy cập trên cả Mac lẫn Windows, Linux. Asset Server đòi hỏi một khoản phí phụ trội là $499 cho mỗi bản copy Unity, và một license Unity Pro.

* 1. NGÔN NGỮ TRONG UNITY

Unity hỗ trợ 2 ngôn ngữ chính là C# và JavaScript.

# Ngôn ngữ C#

* + - 1. *Tổng quan về ngôn ngữ C#*

C# (hay C sharp) là một ngôn ngữ lập trình đơn giản, được phát triển bởi đội ngũ kỹ sư của Microsoft vào năm 2000. C# là ngôn ngữ lập trình hiện đại, hướng đối tượng và được xây dựng trên nền tảng của hai ngôn ngữ mạnh nhất là C++ và Java.

C# làm việc chủ yếu trên bộ khung .NET (.NET framework). Ngôn ngữ lập trình này có khả năng tạo ra nhiều ứng dụng mạnh mẽ và an toàn cho nền tảng Windows. Các thành phần máy chủ, dịch vụ web, ứng dụng di động và nhiều khả năng khác nữa.

C# được thiết kế chủ yếu bởi Anders Hejlsberg kiến trúc sư phần mềm nổi tiếng với các sản phẩm Turbo Pascal, Delphi, J++, WFC.

* + - 1. *Đặc điểm ngôn ngữ*

C#, theo một hướng nào đó, là ngôn ngữ lập trình phản ánh trực tiếp nhất đến .NET Framework mà tất cả các chương trình .NET chạy, và nó phụ thuộc mạnh mẽ vào framework này. Mọi dữ liệu cơ sở đều là đối tượng, được cấp phát và hủy bỏ bởi trình dọn rác Garbage-Collector (GC), và nhiều kiểu trừu tượng khác chẳng hạn như class, delegate, interface, exception... phản ánh rõ ràng những đặc trưng của .NET runtime.

So sánh với C và C++, ngôn ngữ này bị giới hạn và được nâng cao ở một vài đặc điểm nào đó, nhưng không bao gồm các giới hạn sau đây:

Các con trỏ chỉ có thể được sử dụng trong chế độ không an toàn. Hầu hết các đối tượng được tham chiếu an toàn, và các phép tính đều được kiểm tra tràn bộ đệm. Các con trỏ chỉ được sử dụng để gọi các loại kiểu giá trị; còn những đối tượng thuộc bộ gom rác (garbage-collector) thì chỉ được gọi bằng cách tham chiếu.

Các đối tượng không thể được giải phóng tường minh.

Chỉ có đơn kế thừa, nhưng có thể cài đặt nhiều interface trừu tượng (abstract interfaces). Chức năng này làm đơn giản hóa sự thực thi của thời gian thực thi.

C# thì an-toàn-kiểu (typesafe) hơn C++.

Cú pháp khai báo mảng khác nhau("int[] a = new int[5]" thay vì "int a[5]").

Kiểu thứ tự được thay thế bằng tên miền không gian (namespace). C# không có tiêu bản.

Có thêm Properties, các phương pháp có thể gọi các Properties để truy cập dữ liệu.

Có reflection.

* + - 1. *Lịch sử*

Cái tên "C sharp" được lấy cảm hứng từ ký hiệu âm nhạc, trong đó một dấu thăng sau nốt nhạc "#" nghĩa là một nốt được chơi cao hơn nửa cung [13]. Điều này tương tự như trường hợp đặt tên của ngôn ngữ của C++, trong đó "++" chỉ ra rằng giá trị của một biến nên được tăng lên 1. Biểu tượng # cũng giống với bốn ký tự "+" (trong một lưới 2x2), ngụ ý rằng ngôn ngữ là một phiên bản tăng cường của C++ (gấp đôi C++) [14]. Bởi vì giới hạn kỹ thuật của việc hiển thị (các font chuẩn, trình duyệt...) và sự thật là ký tự thăng (U+266F ♯ MUSIC SHARP SIGN (HTML &#9839;)) không hiện diện trong đa số các bàn phím, ký tự (U+0023 # NUMBER SIGN (HTML &#35;)) đã được chọn để diễn đạt một cách tương đương trong cách viết tên ngôn ngữ.

Hậu tố "#" đã được sử dụng bởi một số ngôn ngữ khác của .NET là các biến thể của các ngôn ngữ hiện tại, bao gồm J# (một ngôn ngữ .NET cũng được thiết kế bởi Microsoft có nguồn gốc từ Java 1.1), A# (từ Ada) và ngôn ngữ lập trình chức năng F#.[15] Việc triển khai ban đầu của Eiffel for .NET được gọi là Eiffel#[16]. Hậu tố cũng đã được sử dụng cho các thư viện, chẳng hạn như Gtk# (một wrapper NET cho GTK+ và các thư viện GNOME khác) và Cocoa#.

# Ngôn ngữ Javascript

* + - 1. *Tổng quan về ngôn ngữ Javascript*

JavaScript, theo phiên bản hiện hành, là một ngôn ngữ lập trình kịch bản dựa trên đối tượng được phát triển từ các ý niệm nguyên mẫu. Ngôn ngữ này được dùng rộng rãi cho các trang web, nhưng cũng được dùng để tạo khả năng viết script sử dụng các đối tượng nằm sẵn trong các ứng dụng. Nó vốn được phát triển bởi Brendan Eich tại Hãng truyền thông Netscape với cái tên đầu tiên Mocha, rồi sau đó đổi tên thành LiveScript, và cuối cùng thành JavaScript. Giống Java, JavaScript có cú pháp tương tự C, nhưng nó gần với Self hơn Java. .js là phần mở rộng thường được dùng cho tập tin mã nguồn JavaScript.

Phiên bản mới nhất của JavaScript là phiên bản 1.5, tương ứng với ECMA-262 bản 3. ECMAScript là phiên bản chuẩn hóa của JavaScript. Trình duyệt Mozilla phiên bản 1.8 beta 1 có hỗ trợ không đầy đủ cho E4X - phần mở rộng cho JavaScript hỗ trợ làm việc với XML, được chuẩn hóa trong ECMA-357.

* + - 1. *Ứng dụng*

JavaScript là một ngôn ngữ lập trình dựa trên nguyên mẫu với cú pháp phát triển từ C. Giống như C, JavaScript có khái niệm từ khóa, do đó, JavaScript gần như không thể được mở rộng.

Cũng giống như C, JavaScript không có bộ xử lý xuất/nhập (input/output) riêng. Trong khi C sử dụng thư viện xuất/nhập chuẩn, JavaScript dựa vào phần mềm ngôn ngữ được gắn vào để thực hiện xuất/nhập. Trên trình duyệt, rất nhiều trang web sử dụng JavaScript để thiết kế trang web động và một số hiệu ứng hình ảnh thông qua DOM. JavaScript được dùng để thực hiện một số tác vụ không thể thực hiện được với chỉ HTML như kiểm tra thông tin nhập vào, tự động thay đổi hình ảnh,... Ở Việt Nam, JavaScript còn được ứng dụng để làm bộ gõ tiếng Việt giống như bộ gõ hiện đang sử dụng trên trang Wikipedia tiếng Việt. Tuy nhiên, mỗi trình duyệt áp dụng JavaScript khác nhau và không tuân theo chuẩn W3C DOM, do đó trong rất nhiều trường hợp lập trình viên phải viết nhiều phiên bản của cùng một đoạn mã nguồn để có thể hoạt động trên nhiều trình duyệt. Một số công nghệ nổi bật dòng JavaScript để tương tác với DOM bao gồm DHTML, Ajax và SPA.

Bên ngoài trình duyệt, JavaScript có thể được sử dụng trong tập tin PDF của Adobe Acrobat và Adobe Reader. Điều khiển Dashboard trên hệ điều hành Mac OS X phiên bản 10.4 cũng có sử dụng JavaScript. Công nghệ kịch bản linh động (active scripting) của Microsoft có hỗ trợ ngôn ngữ JScript làm một ngôn ngữ kịch bản dùng cho hệ điều hành. JScript.NET là một ngôn ngữ tương thích với CLI gần giống JScript nhưng có thêm nhiều tính năng lập trình hướng đối tượng.

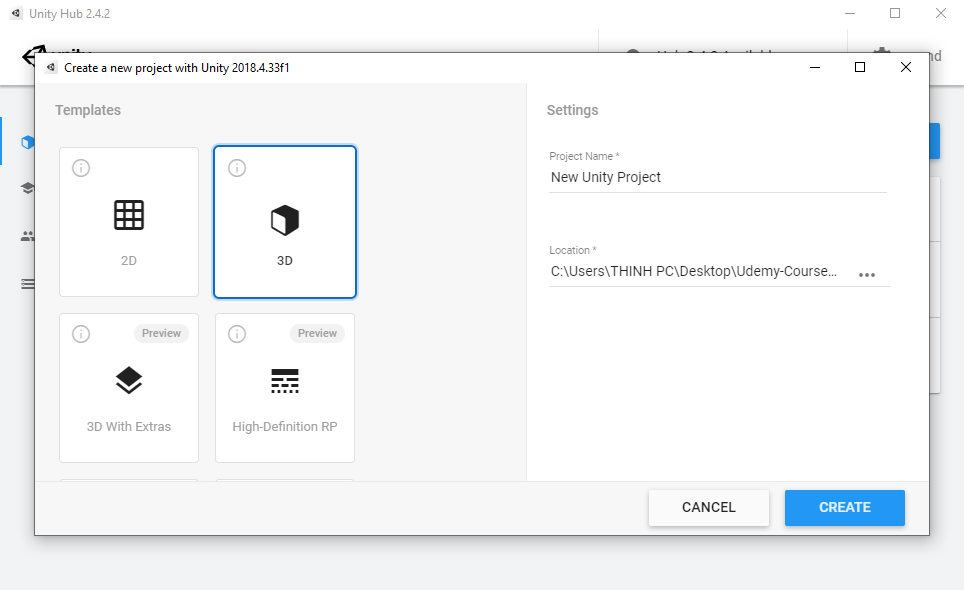
Mỗi ứng dụng này đều cung cấp mô hình đối tượng riêng cho phép tương tác với môi trường chủ, với phần lõi là ngôn ngữ lập trình JavaScript gần như giống nhau.

# CHƯƠNG 2 LẬP TRÌNH VỚI UNITY

* 1. GIAO DIỆN UNITY

# Khởi tạo project

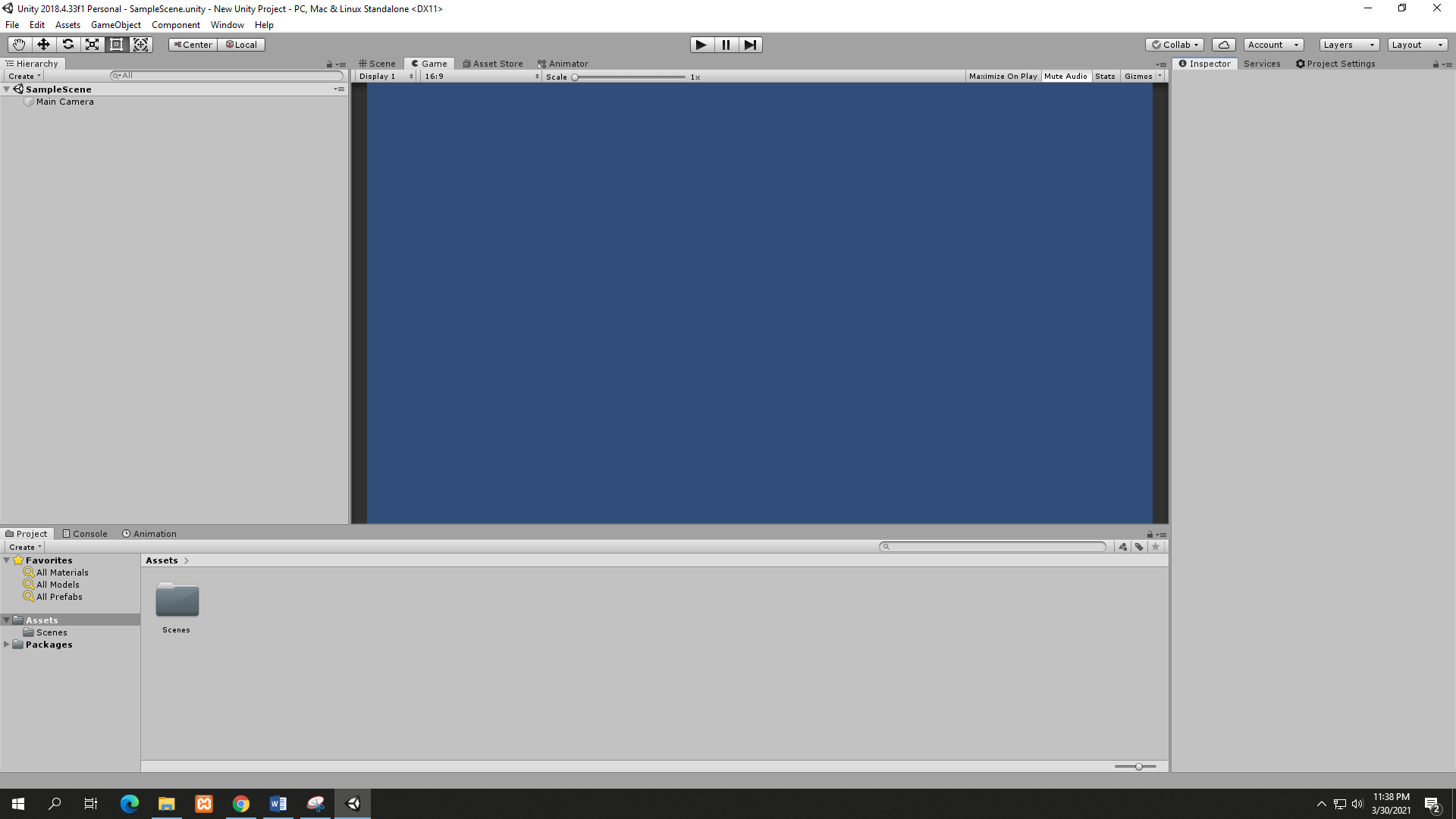
Việc khởi tạo một project trong Unity tương đối đơn giản và không gây khó khăn cho người dùng. Ta có thể chọn một khởi tạo project 2D hoặc 3D và thêm một số Asset Packages cần thiết .



Hình 2.1 Khởi tạo project (bản 2018.4.33f1)

# Giao diện chính

Sau khi vào project mới tạo , giao diện làm việc của Unity như sau :



Hình 2.2 Giao diện chính của Unity

# Thanh công cụ

Thanh công cụ của Unity bao gồm 5 thành phần. Mỗi phần có ý nghĩa và mục đích khác nhau.

# C:\Users\L\Desktop\Luanvan\image\2.2.PNGTab Scene

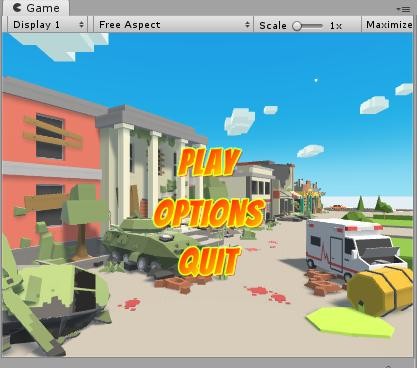
Hình 2.3 Tab Scene

Tab Scene là nơi bố trí các Game Object như cây cối, cảnh quan, enemy, player, camera, … trong game. Sự bố trí hoạt cảnh là một trong những chức năng quan trọng nhất của Unity.

# Gizmo

Nếu ta xây dựng game 3D, ở góc trên bên phải của khung Scene sẽ xuất hiện biểu tượng Gizmo (hệ trục toạ độ Oxyz). Đây là nơi hiển thị góc nhìn hiện tại của khung Scene, giúp ta thay đổi góc nhìn một cách nhanh chóng. Mỗi màu sắc thể hiện một trục toạ độ khác nhau , ta có thể click chuột vào các trục để thay đổi góc nhìn tương ứng với trục đó.

# Tab Game



Hình 2.4 Tab game

game.

Đây là màn hình hiển thị demo game, là góc nhìn từ các camera trong

Thanh công cụ trong cửa sổ game cung cấp các tuỳ chỉnh về độ phân

giải màn hình, tỷ lệ màn hình , phóng to , thu nhỏ , tùy chọn toàn màn hình , bật tắt âm , thông số (stats), gizmos, tuỳ chọn bật/tắt các component, …

# Tab Project



Hình 2.5 Tab Project

Đây là cửa sổ explorer của Unity, hiển thị thông tin của toàn bộ các tài nguyên (assets) trong game.

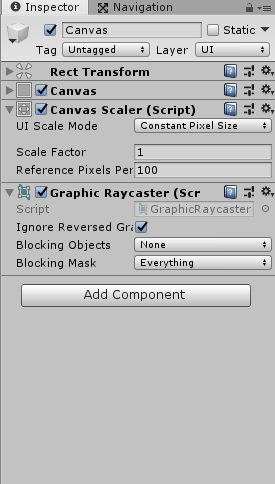
Cột bên trái hiển thị assets và các mục yêu thích dưới dạng cây thư mục tương tự như Windows Explorer. Khi click vào một nhánh trên cây thư mục thì toàn bộ nội dung của nhánh đó sẽ được hiển thị ở khung bên phải. Ta có thể tạo ra các thư mục mới bằng cách Right click -> Create -> Folder hoặc nhấn vào nút Create ở góc trên bên trái cửa sổ Project và chọn Folder. Các tài nguyên trong game cũng có thể được tạo ra bằng cách này.

Phía trên cây thư mục là mục Favorites, giúp chúng ta truy cập nhanh vào những tài nguyên thường sử dụng. Chúng ta có thể đưa các tài nguyên vào Favorites bằng thao tác kéo thả.

Chúng ta có thể dễ dàng tiếp cận các thư mục con hoặc thư mục gốc bằng cách click chuột vào mũi tên hoặc tên thư mục.

Khi game càng lớn, số lượng assets càng nhiều, thanh tìm kiếm sẽ là công cụ cần thiết để xác định asset cần tìm.

# Tab Inspector



Hình 2.6 Tab Inspector

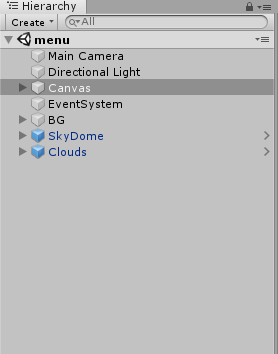
Tab Inspector hiển thị chi tiết các thông tin về Game Object đang làm việc, kể cả những component được đính kèm và thuộc tính của nó. Chúng tacó thể điều chỉnh, thiết lập mọi thông số và chức năng của Game Object thông qua cửa sổ Inspector.

Mọi thuộc tính thể hiện trong Inspector đều có thể dễ dàng tuỳ chỉnh trực tiếp mà không cần thông qua một kịch bản định trước. Tuy nhiên Scripting API cung cấp một số lượng nhiều và đầy đủ hơn do giao diện Inspector là có giới hạn.

Các thiết lập của từng component được đặt trong menu. Ta có thể click chuột phải, hoặc chọn icon hình bánh răng nhỏ để xuất hiện menu này.

Ngoài ra Inspector cũng thể hiện mọi thông số Import Setting của asset đang làm việc như hiển thị mã nguồn của Script, các thông số animation, …

# Tab Hierarchy



Hình 2.7 Tab Hierachy

Đây là tab hiển thị những Game Object được đưa và sử dụng tại tabScene, hiển thị tại đây sẽ giúp ta dễ dàng xem , chỉnh sửa , thêm mới hoặc xóa bỏ

* 1. GAME OBJECT

# Các thuộc tính và hàm

* + - 1. *Transform*

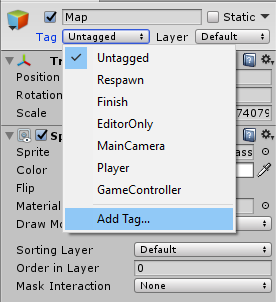
Thành phần Transform là một thành phần bắt buộc, không thể thêm hoặc xoá đối với bất kỳ GameObject nào. Transform lưu trữ các thông tin về vị trí, phép quay và tỉ lệ của đối tượng. Ngoài ra Transform còn có một số thuộc tính và phương thức hay để thao tác với GameObject.

* + - 1. *tag-name*

Unity sử dụng hai cơ chế để lưu trữ và quản lý danh sách các GameObject, được hiển thị trong cửa sổ Hierarchy. Thuộc tính name được kế thừa lại từ lớp Object, lưu trữ tên của đối tượng được hiển thị ở Hierarchy.

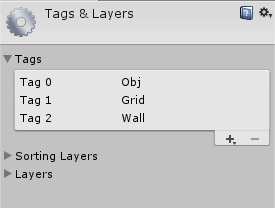
Thuộc tính tag được sử dụng để phân loại và định danh các GameObject, phải được định nghĩa sẵn trước khi sử dụng. Các đối tượng chưa

được phân loại sẽ được gắn mặc định tag “Untagged”. Để định nghĩa tag, ta có thể sử dụng Tag Manager tại cửa sổ Inspector của bất kỳ đối tượng nào.



Hình 2.8 Danh sách các tag

Chọn nút Add Tag, bấm chọn nút dấu + để thêm một tag mới và đặt tên cho nó:



Hình 2.9 Thêm tag

* + - 1. *Layer*

Hình 2.10 Layer

Tại Tag Manager, chúng ta cũng có thể tạo ra các Layer và Sorting Layer để thuận tiện trong việc quản lý các đối tượng. Sorting Layer được sử dụng để phân lớp hiển thị các đối tượng và sử dụng với thành phần SpriteRenderer.

Layer được sử dụng để phân lớp đối tượng và tuỳ chọn hiển thị trong Camera. Một số lớp được Unity tạo sẵn cơ bản là đủ với các nhu cầu thông thường của lập trình viên.

* + - 1. *Active*

Hai thuộc tính activeInhierarchy và activeSelf được sử dụng để kiểm tra trạng thái hoạt động của đối tượng. Đối với việc thiết lập trạng thái, ta sử dụng hàm SetActive với giá trị truyền vào là true/false, là trạng thái cần thiết để thiết lập cho đối tượng đó.

Nếu đối tượng cha có trạng thái active là false thì dù ta có thay đổi trạng thái của đối tượng con, chúng vẫn sẽ trả về giá trị false.

* + - 1. *isStatic*

Một số đối tượng trong game, ví dụ như bản đồ hay các chướng ngại vật tĩnh đều không có khả năng di chuyển. Do đó chúng ta nên thiết lập trạng thái Static cho các đối tượng đó để tiết kiệm tài nguyên. Các đối tượng Static sẽ được tiền xử lý và gom chung thành một đối tượng gọi là Batching object,

nhờ đó sẽ tiết kiệm được một số lượng Draw call (Draw call là số lần render một đối tượng, mỗi lần render là một draw call).



Hình 2.11 Môi trường được tích Static

Tương tự như trạng thái Active của đối tượng, trong Unity Editor có 1 checkbox dùng cho việc thiết lập trạng thái static của đối tượng. Thuộc tính isStatic dùng để kiểm tra trạng thái tĩnh của đối tượng, từ đó bỏ qua các thao tác xử lý với đối tượng này.

* + - 1. *Tìm kiếm GameObject a.Tìm kiếm với name*

Hàm Find được Unity thiết kế để tìm kiếm một đối tượng trong cửa sổ Hierarchy. Cú pháp của hàm như sau:

public static GameObject Find(string name);

Hàm sẽ trả về giá trị GameObject là chính đối tượng được tìm thấy, nếu không có đối tượng có tên cần tìm, hàm sẽ trả về null.

Chuỗi name truyền vào chính là tên của đối tượng cần tìm. Nếu chuỗi kí tự có chứa kí hiệu "/", Unity sẽ xem đó là đường dẫn của đối tượng ở trong cửa sổ Hierarchy.

Lưu ý: Hàm sẽ chỉ trả về đối tượng được active trong cửa sổ Hierarchy. Mọi đối tượng khác cho dù có tên đúng yêu cầu nhưng chưa được active cũng sẽ bị bỏ qua.

*b.Tìm kiếm với tag*

Để tìm kiếm với tag, ta sử dụng hàm FindGameObjectWithTag hoặc FindGameObjectsWithTag tương ứng để tìm một hay toàn bộ các đối tượng có tag cần tìm. Cách sử dụng tương tự như hàm Find đề cập ở trên, sử dụng tag để tìm kiếm đối tượng.

Ví dụ :

GameObject stdioObject = GameObject.FindGameObjectWithTag("StdioObject");

* + - 1. *Khởi tạo Game Object*
         1. *Khởi tạo Game Object với hàm CreatePrimitive*

Khởi tạo một đối tượng được định nghĩa sẵn bởi Unity. Đối tượng được tạo ra sẽ có sẵn Mesh Renderer và Collider tương ứng với hình dạng của đối tượng.

Ví dụ cách sử dụng hàm như sau:

// khởi tạo 1 plane được định nghĩa sẵn trong Unity

GameObject plane = GameObject.CreatePrimitive(PrimitiveType.Plane);

// khởi tạo 1 cube được định nghĩa sẵn trong Unity sau đó gán tọa độ cho nó

GameObject cube = GameObject.CreatePrimitive(PrimitiveType.Cube);

cube.transform.position = new Vector3(0, 0.5f, 0);

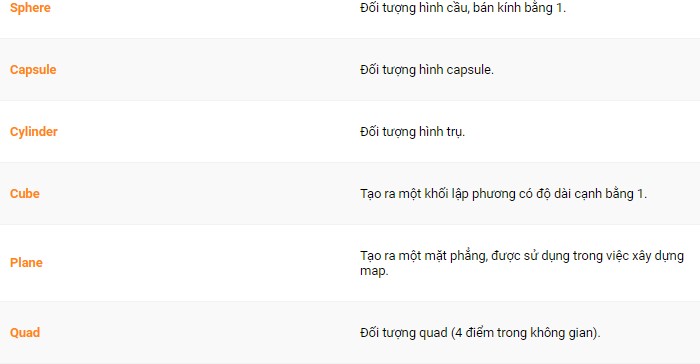
// khởi tạo 1 sphere được định nghĩa sẵn trong Unity sau đó gán tọa độ cho nó

GameObject sphere = GameObject.CreatePrimitive(PrimitiveType.Sphere);

sphere.transform.position = new Vector3(0, 1.5f, 0);

Danh sách các PrimitiveType được hỗ trợ:

1



Hình 2.12 Danh sách các PrimitiveType được hỗ trợ

* + - * 1. *Khởi tạo Game Object vớihàm Instantiate*

Instantiate được sử dụng để khởi tạo một đối tượng được lập trình viên xây dựng sẵn. Nguyên mẫu hàm như sau:

public static Object Instantiate(Object original, Vector3 position, Quaternion rotation);

public static Object Instantiate(Object original);

Nếu không truyền vào position và rotation, hàm sẽ lấy các giá trị của đối tượng cơ sở để gán cho đối tượng mới, hoặc trả về Vector3.zero và Quaternion.identity nếu không khả dụng.



Hình 2.13 Dùng hàm Instantiate để tạo ra các Zombie đã được đặt các thuộc tính sẵn

Khi sử dụng Instantiate, toàn bộ các đối tượng con và cấu trúc sẽ được sao chép với đúng tính chất của chúng. Tuy nhiên, đối tượng cha (nếu có) sẽ được đặt là null. Do đó đối tượng được clone có thể sẽ không nằm chung vị trí với đối tượng ban đầu.

* + - 1. *Giải phóng Game Object*
         1. *Giải phóng Game Object với hàmDestroy()*

Hàm Destroy được sử dụng để giải phóng một GameObject, component hay tài nguyên được sử dụng trong game. Hàm nhận vào giá trị là đối tượng cần giải phóng, cùng với thời gian delay (mặc định là 0). Việc huỷ đối tượng sẽ được delay cho đến khi kết thúc hàm Update, nhưng đối tượng sẽ được huỷ trước quá trình render.

Dưới đây là ví dụ để ta hiểu về chức năng và cách sử dụng hàm Destroy:

// Giải phóng đối tượng game Object

Destroy (gameObject);

// Giải phóng script đang được gán hiện tại

Destroy (this);

// Giải phóng Component BoxCollider được gắn vào trong Unity

Destroy (GetComponent<BoxCollider>());

// Giải phóng gameObject sau 5 giây

Destroy (gameObject, 5);

* + - * 1. *Hàm DontDestroyOnLoad()*

Khi thực hiện chuyển scene, các đối tượng ở scene cũ sẽ được giải phóng hoàn toàn. Hàm này giúp chỉ định một đối tượng nào đó sẽ không bị giải phóng khi chuyển scene. Cách sử dụng hàm như sau:

DontDestroyOnLoad(targetObject);

targetObject chính là đối tượng sẽ được giữ lại khi chuyển scene. Đối tượng này vẫn có thể được giải phóng bằng hàm Destroy.

# Thao tác với Component

Một GameObject sẽ có nhiều thành phần cấu tạo nên nó như là hình ảnh (sprite render), tập hợp các hành động (animator), thành phần xử lý va chạm (collision), tính toán vật lý (physical), mã điều khiển (script), các thành phần khác...; mỗi thứ như vậy gọi là một component của GameObject. Dưới đây sẽ là một số thao tác với Component

* + - 1. *GetComponent và GetComponents*

Hàm GetComponent sẽ trả về component đầu tiên cần tìm của đối tượng, trong khi GetComponents sẽ tra về một danh sách toàn bộ các component cần tìm. Nếu không tìm được sẽ trả về null hoặc mảng không có phần tử tương ứng.

Nguyên mẫu của hàm như sau:

* + - * 1. public Component GetComponent(Type type);
        2. public Component GetComponent(string type);

Hàm có thể nhận vào giá trị kiểu Type hoặc kiểu string đều được chấp nhận. Tuy nhiên, để tối ưu ta nên sử dụng kiểu Type trong hầu hết mọi trường hợp. Một cú pháp khác để lấy một thành phần với Type như sau:

public Type GetComponent<Type>();

Cú pháp này sử dụng template để lấy ra các component. Đây là cách thông dụng nhất được lập trình viên sử dụng trong C# Script.

Một trường hợp bắt buộc phải sử dụng string để tìm kiếm là khi cần truy xuất các lớp được định nghĩa trong C# Script bằng Javascript hoặc ngược lại. Do tính đa ngôn ngữ, Unity cho phép lập trình viên sử dụng C# hoặc Javascript để lập trình và các ngôn ngữ này không thể được liên kết trực tiếp bằng Type. Lúc này ta sẽ sử dụng string và thao tác tương tự như sau:

comp = gameObject.GetComponent("Comp") as Comp; comp.xyz = 1.0f;

* + - 1. *Tìm kiếm Component trong các đối tượng cha hoặc con*

Unity cũng cung cấp sẵn một số phương thức giúp ta có thể thao tác được với component thuộc đối tượng cha/con của nó. Cách thức hoạt động tương tự như GetComponent và GetComponents.

Các hàm như sau:

* + - * 1. GetComponentInChildren.
        2. GetComponentsInChildren.
        3. GetComponentInParent.
        4. GetComponentsInParent.

Các hàm này chỉ có thể được sử dụng với Type của component, kiểu string không được chấp nhận.

* 1. CƠ CHẾ VẬT LÝ TRONG UNITY

Nói đến game không thể không nói đến tương tác vật lý tồn tại tại đa số game, tương tác vật lý áp dụng các luật vật lý khiến game có độ thực tế cao hơn. Mục đích của tương tác vật lý là sử dung thiết bị vi tính để tái hiện lại những hiện tượng vật lý qua các công thức, tại thời điểm hiện tại máy tính cũng như đa số thiết bị điện tử đều có phần cứng khá tốt để tạo các tương tác gần giống tương đương với thế giới thực.

Unity hỗ trợ cơ chế vật lý với 1 số các component chính sau :

# Rigidbody

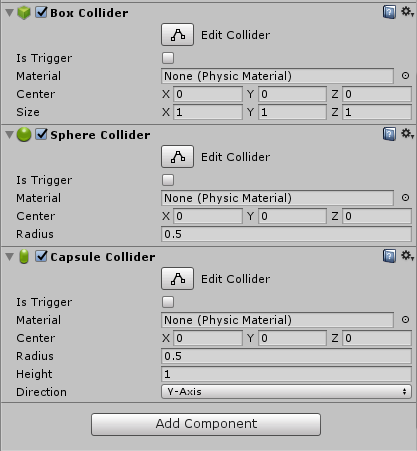
Hình 2.14 Rigidbody

Rigidbody là bộ phận quan trọng nhất trong engine vật lý, mỗi object đều tạo ra bởi Unity đều được mặc định gắn componenet Rigidbody, nếu tạo ra object rỗng để đưa rigidbody vào object trong Unity chỉ cần chọn object mình muốn, chọn Components, Physics và Rigidbody. Khi áp dụng rigidbody lên bất kì vật nào, vật đó sẽ tự động có những tương tác vật lý mà không cần dòng code nào cả, rigidbody sẽ được kéo xuống bởi trọng lực và sẽ va chạm với các vật chạm vào nó nếu component Collider tồn tại với cả 2 object. RigidBody có những thuộc tính sau đây:

* Mass – Khối lượng của vật, không nên đặt chỉ số này cao hơn hoặc thấp hơn 100 lần so với các rigidbody khác.
* Drag – Sức cản không khí sẽ ảnh hưởng đến object thế nào, 0 nghĩa là hoàn toàn không có sức cản, vô tận sẽ khiến cho object ngừng di chuyển.
* Angular Drag – Sức cản không khỉ khi vật quay, lưu ý là không thể khiến object ngừng quay với angular drag vô tận.
* Use Gravity – Nếu check, trọng lực sẽ được áp dụng lên Game Object.
* Is Kinematic – Nếu check, Game Object sẽ không được điều khiển bởi engine vật lý mà chỉ có thể điều khiển bởi transform.
* Interpolate – Dùng để điều chỉnh sự va chạm, độ va chạm có thể nhạy hơn tùy từng trường hợp.
* Collision Detection – Dùng để ngăn chặn các Game Object di chuyển quá nhanh xuyên qua các Game Object khác mà không bị va chạm, như khi viên đạn di chuyển nhanh quá, và vượt qua Game Object khác trước khi va chạm được update.
* Constraints : Dùng để ngăn Game Object di chuyển theo trục x,y,z hoặc ngăn Game Object quay theo trục x,y,z
* Ngoài ra RigidBody có thể được áp dụng lực trong code với function AddForce()

# Colliders

Colliders là vật mà engine vật lý sẽ dùng để có thể nhận ra sự va chạm, không giống như các mesh, chúng nhận biết được mỗi khi va chạm với nhau. Đa số collider có hình dạng đơn giản nhằm mục đích tính toán đơn giản và dễ dàng hơn, phần lớn các object trong Unity sẽ được gắn collider mỗi khi tạo ra, với Cube là Box Collider, Sphere là Sphere Collider, Cylinder là Capsule Collider.



Hình 2.15 Các loại Colliders

Unity cung cấp những API sau đây để phát hiện sự va chạm của các collider.

* void OnCollisionEnter(Collision collision) – Chạy 1 lần tại thời điểm va chạm giữa 2 vật.
* void OnCollisionStay(Collision collision) – Chạy trong mỗi khung hình tại thời điểm 2 vật còn chạm vào nhau.
* void OnCollisionExit(Collision collision) – Chạy tại khung hình cuối cùng khi 2 vật không còn chạm vào nhau nữa. Với class Collision chúng ta có thể lấy ra những thuộc tính như:
* Contacts – điểm va chạm giữa 2 vât, tính bằng vector3.
* GameObject – game object va chạm với object gốc.
* RelativeVelocity – vận tốc tương đối.

Một ví dụ đơn giản để áp dụng những thông tin được API của Unity cung cấp, khi nhân vật Assassin rơi từ trên độ cao xuống dưới mặt đất, số lượng sát thương sẽ được áp dụng vào nhân vật Assassin:

void OnCollisionEnter(Collision collision)

{

if (collision.relativeVelocity.magnitude < 10)

{ //Áp dụng sát thương nếu vận tốc thấp.

Assassin.ApplyDamage(collision.relativeVeloc ity.magnitude \* damagePerVelocity);

return;

}

if (collision.relativeVelocity.magnitude >= 10)

{ //Thật đáng tiếc Assassin.Kill();

return;

}

}

Ngoài ra trigger là 1 lựa chọn rất hữu dụng trong Collider . Để có thể dùng Trigger trong collider, đơn giản chỉ cần tick vào checkbox As Trigger trong Componenet Collider ở Inspecter. Khi trigger được chọn collider sẽ không bị va chạm bởi bất kỳ object nào, nhưng bản thân sẽ được dùng để phát hiện những va chạm trên nó, tạo ra những event có thể điều khiển được trong code với những function sau đây:

* void OnTriggerEnter(Collider other) – Chạy 1 lần tại thời điểm va chạm giữa 2 vật.
* void OnTriggerStay(Collider other) – Chạy trong mỗi khung hình tại thời điểm 2 vật còn chạm vào nhau.
* void OnTriggerExit(Collider other) – Chạy tại khung hình cuối cùng khi 2 vật không còn chạm vào nhau nữa.

Các class này sẽ đươc gọi ra khi một Collider khác va chạm vào Collider có thuộc tính As Trigger, từ Collider có thể lấy ra thông tin về object bị va chạm, cụ thể có thể dùng nó để kiểm tra sự va chạm vào 1 vật mà không áp dụng lại lực tương đương, như ví dụ về nhân vật người chơi khi chạm vào Zombie sẽ mất máu

void OnTriggerEnter(Collider other) {

//Nếu vật va chạm có tên Player if (other.name == “Player”) {

//Trừ máu GameController.TakeDamage();

}

}

* 1. LỚP INPUT

Input là một thành phần quan trọng trong bất kì ứng dụng hoặc game nào. Với Unity, Input cũng quan trọng và là thành phần không thể thiếu. Dưới đây sẽ là 1 số hàm ,sự kiện chính của Input trong Unity

# Sự kiện OnMouse

Đây là một sự kiện cơ bản thường được sử dụng trong Unity cho phép lập trình viên dùng chuột máy tính để tương tác với các đối tượng trong game. Giả sử trong game ta có một GameObject, ví dụ như một quả boom. Chúng ta mong muốn khi người chơi click chuột vào nó thì quả boom đó sẽ phát nổ. Để làm được điều này, ta sử dụng hàm OnMouseDown được hiện thực sẵn bởi Unity. Hàm này sẽ được tự động gọi mỗi khi người dùng click vào đối tượng. Vậy công việc tới đây khá đơn giản, ta chỉ cần hiện thực một hàm thể hiện một vụ nổ và trong hàm OnMouseDown, ta sẽ có một lời gọi đến hàm này.

Ngoài OnMouseDown ta còn có thêm một số hàm liên quan:

* OnMouseUp: xảy ra khi người dùng thả chuột khi đã click chuột trước đó.
* OnMouseEnter: xảy ra khi người dùng di chuyển chuột đi vào phạm vi Collider của GameObject
* OnMouseExit: xảy ra khi người dùng di chuyển chuột ra khỏi phạm vi Collider của GameObject.
* Cách sử dụng các hàm này tương tự với OnMouseDown.

# Sự kiện GetMouseButton

Các sự kiện OnMouse làm việc khá hiệu quả và thường được sử dụng khi ta cần phát hiện và xử lý khi người dùng click chuột. Tuy nhiên nó có một số điểm còn hạn chế

Trở về ví dụ quả boom ở trên. Giả sử bây giờ ta muốn dùng chuột để di chuyển quả boom đi một vị trí khác và sau đó mới kích nổ nó. Rõ ràng ở đây ta gặp một vấn đề! Hàm OnMouseDown sẽ được gọi khi Mouse click xuống mà không cần biết đó là Mouse left hay Mouse right. Quả boom sẽ phát nổ ngay khi vừa click vào mà chưa kịp di chuyển.

Vậy bây giờ ta muốn sử dụng chuột trái để di chuyển và chuột phải để kích hoạt thì sao?

Điều đó có thể thực hiện dễ dàng bằng cách sử dụng các hàm Input.GetMouseButton(x). Cụ thể :

Hàm Input.GetMouseButton(0) đại diện cho Mouse Left. Hàm Input.GetMouseButton(1) đại diện cho Mouse Center. Hàm Input.GetMouseButton(2) đại diện cho Mouse Right.

Các hàm trên cần phải được gọi liên tục sau mỗi frame để sự kiện được kiểm tra một cách chính xác nhất.

# Nhập dữ liệu từ bàn phím

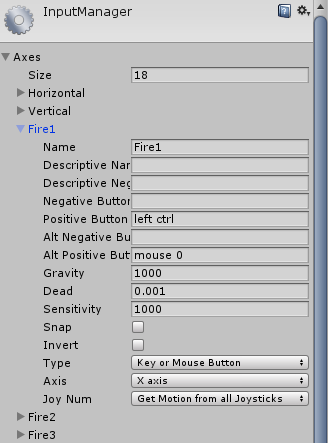
Đây là một dạng input thường xuyên được sử dụng khi lập trình các game trên Windows, là nơi sử dụng bàn phím vật lý. Những hành động thường gặp là chạy, nhảy, tấn công đối thủ sẽ được gắn vào các phím trên keyboard. Để lấy input từ bàn phím ta có 2 cách: GetKey(hay GetButton) và GetAxis

* + - 1. *GetKey và GetButton*

Trong Unity, GetKey hay GetButton là cách mà ta sẽ nhận input từ bàn phím thông qua class Input. Điểm khác biệt giữa hai cái này là GetKey sử dụng KeyCode được quy định sẵn trong Unity. Mỗi KeyCode đại diện cho một phím bấm.

GetButton sẽ sử dụng tên do chính lập trình viên tự quy định để đại diện cho một phím bất kì. Chẳng hạn có thể đặt “DauCach” đại diện cho phím Space. Về cơ bản lúc này dấu cách tương đương với Key.Space

Để thêm một tên đại diện cho một phím bất kì ta có thể vào Edit → Project Settings → Input menu. Với Name là tên thay thế cho phím và PostiveButton chính là phím cần đại diện.



Hình 2.16 Cài đặt Input trong Unity

* + - 1. *GetAxis*

Điểm khác biệt giữa GetAxis so với các phương thức GetKey hay GetButton là nó không trả về giá trị true/false như thông thường mà sẽ trả về một giá trị float có phạm vi nằm trong khoảng [-1,1] phụ thuộc vào thời gian nhấn giữ của một phím.

GetAxis thường làm việc cùng lúc với 2 phím đồng thời được khai báo cụ thể trong Input Manager. Một phím gọi là Negative Button, phím còn lại gọi là Positive Button. có giá trị lần lượt là -1 và 1. Ở trạng thái bình thường, cả 2 phím này không được nhấn hàm sẽ trả về giá trị 0. Khi trạng thái của phím PositiveButton thay đổi, hàm sẽ tăng giá trị từ 0 lên đến 1. Khi nhả phím ra nó sẽ giảm giá trị trở về 0. Tương tự như vậy với Negative Button với khoảng [-1, 0].

Thời gian thay đổi các cặp giá trị [0, 1] [1,0] , [0, -1] [-1,0] có thể được điều chỉnh bằng thuộc tính Gravaity thông qua Input Manager. Ta có thể thiết lập thời gian từ 0 đến 1 sẽ tăng dần đều, từ 1 trở về 0 sẽ giảm dần đều... để phù hợp với từng trường hợp cụ thể của Game.

GetAxis được sử dụng rất nhiều trong việc di chuyển Player trong game theo thời gian thực. Giá trị của hàm phụ thuộc vào thời gian nhấn giữ phím do đó có thể dùng để di chuyển nhân vật nhanh/chậm theo ý muốn dễ dàng.

* 1. CÁC HỆ TỌA ĐỘ TRONG UNITY

# World Coordinate

Là hệ toạ độ của thế giới thực trong game. Hệ toạ độ này tuân theo quy tắc bàn tay trái:

* Chiều x dương là bên phải.
* Chiều y dương hướng lên trên.
* Chiều z dương hướng vào bên trong màn hình.

Toạ độ của đối tượng được thể hiện trong thành phần Transform chính là toạ độ trong World coordinate. Các đối tượng con còn có một toạ độ khác được lưu trữ tại các thuộc tính localPosition, localRotation và localScale của Transform.

# Viewport Coordinate

Là hệ toạ độ của camera. Trong Unity, các trục toạ độ của World space và Viewport space trùng nhau, nhưng khác nhau về đơn vị của các trục. Trong camera, góc dưới bên trái có toạ độ (0, 0), góc trên bên phải có toạ độ (1, 1).

Các giá trị này có thể được chỉnh sửa thông qua Viewport Rect trong Inspector của đối tượng Camera.

# Screen Coordinate

Screen Coordinate là hệ toạ độ của màn hình hiển thị. Đơn vị của các trục là pixel, góc dưới bên trái màn hình có toạ độ (0, 0) trong khi góc trên bên phải là (screenWidth – 1, screenHeight – 1).

# UI Coordinate

Là hệ toạ độ của các đối tượng UI, giá trị của các trục toạ độ sẽ nằm trong khoảng (0, 1). Góc trên bên trái có toạ độ (0, 0) và góc dưới bên phải là (1, 1).

# Chuyển đổi giữa các hệ toạ độ trong Unity

Trong khi thao tác với các đối tượng thông qua script, nhiều trường hợp ta nhận được input từ các sự kiện click chuột, touch, … và dùng nó để thao tác với các đối tượng. Thông thường các sự kiện này sẽ trả về toạ độ của Screen space. Vậy làm sao chúng ta có thể chuyển đổi sang World space để thao tác được với các đối tượng một cách chính xác?

Unity hỗ trợ sẵn cho chúng ta các hàm chuyển đổi giữa các Coordinate system. Các hàm này được gắn vào camera chính đang được sử dụng (trong trường hợp game có nhiều hơn 1 camera):

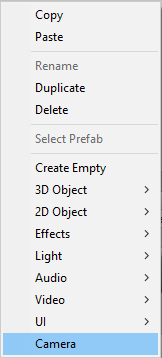
* Camera.WorldToScreenPoint
* Camera.WorldToViewportPoint
* Camera.ScreenToViewportPoint
* Camera.ScreenToWorldPoint
* Camera.ViewportToScreenPoint
* Camera.ViewportToWorldPoint

Các hàm trên sẽ nhận vào một Vector3 và trả về một Vector3 tương ứng với hệ toạ độ cần chuyển.

Đối với UI Coordinate, Unity hỗ trợ chuyển từ UI sang Screen Coordinate và ngược lại. Do đó chúng ta có thể sẽ cần thực hiện hai thao tác liên tiếp để thực hiện được điều cần thiết. Các hàm đó như sau:

* GUIUtility.ScreenToGUIPoint
* GUIUtility.GUIToScreenPoint
  1. CAMERA

Camera là một khái niệm quen thuộc đối với mọi nền tảng phát triển game. Với các thao tác chỉnh sửa và chuyển đổi camera, chúng ta có thể tạo ra vô số hiệu ứng độc đáo và ứng dụng vào game một cách sinh động và hấp dẫn, thu hút nhiều người chơi. Để tạo mới 1 đối tượng Camera trong Unity , ta nhấp chuột phải trong tab Hierarchy và chọn Camera



Hình 2.17 Tạo đối tượng Camera

Sau đây sẽ là 1 số thuộc tính chính của Camera trong Unity

# Projection

Là góc chiếu của camera. Đối với các loại camera trong game thì có hai loại phép chiếu:

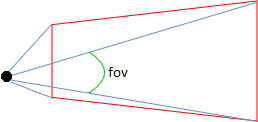
Phép chiếu song song – Orthographic. Phép chiếu xa gần – Perspective.

Orthographic được sử dụng trong các game 2D, các đối tượng sẽ được vẽ đúng tỉ lệ dù ở bất kỳ khoảng cách nào của chiều sâu.

Perspective được ứng dụng trong game 3D, mô phỏng lại thế giới thực của con người. Các vật thể càng ở xa thì tỉ lệ hình ảnh sẽ càng nhỏ và ngược lại.

# Field of view (fov)

Là góc mở của camera Perspective. Đối với Orthographic camera, tham số này sẽ được bỏ qua. Fov được mô tả thông qua hình minh hoạ sau:



Hình 2.18 Minh họa Field of view (fov)

Với cùng một đối tượng và một vị trí, nếu fov càng lớn, đối tượng sẽ càng nhỏ và ngược lại. Đây là một đặc điểm thú vị và được ứng dụng trong việc tạo hiệu ứng zoom trong môi trường 3D.

# Tùy chỉnh Camera tại tab Inspector

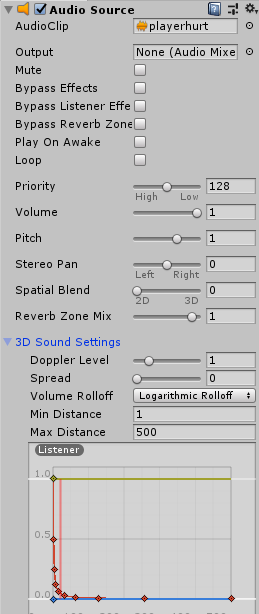
Hình 2.19 Tùy chỉnh camera trong Unity

Tại Inspector, chúng ta có thể chỉnh sửa các thuộc tính minh hoạ ở trong hình. Chi tiết một số thuộc tính thường dùng như sau:

* Clear Flag: Quy định cách thức xoá các đối tượng. Trong 3D game tuỳ chọn Skybox thường được sử dụng và tuỳ chọn Solid color đối với game 2D.
* Background: Màu nền sau khi các đối tượng đã được render.
* Culling Mask: Quy định các layer sẽ được vẽ hoặc ẩn đi.
* Projection: Chọn phép chiếu cho camera.
* Size: Kích thước của camera. Tuỳ chọn này chỉ có khi phép chiếu được chọn là Orthographic.
* Field of view: Góc đo tính bằng độ, là góc đo chiều rộng góc nhìn của Camera.
* Clipping Planes: Bao gồm hai thuộc tinh là Near và Far, là khung nhìn thấy của camera.
* Viewport Rect: Kích thước của camera và vị trí của nó trên màn hình. Các giá trị này sử dụng hệ toạ độ Screen Coordinate (giới thiệu ở phần dưới).
  1. ÂM THANH TRONG UNITY

# Audio Source Component

Audio Source là một built-in component được Unity xây dựng sẵn để hỗ trợ phát các file âm thanh trong game. Để sử dụng Audio Source, ta chọn Add Component → Audio → Audio Source. Các thuộc tính của Audio Source được hiển thị trong hình sau:



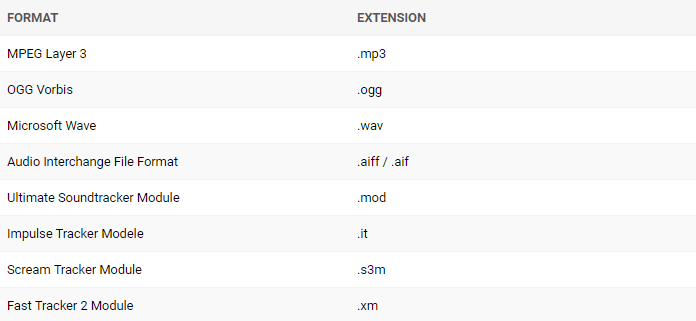
Hình 2.20 Tùy chỉnh Audio trong Unity Các chức năng chính:

|  |  |
| --- | --- |
| **Thuộc tính** | **Ý nghĩa** |
| Audio Clip | File âm thanh sẽ được gán vào AudioClip để xử lý và phát âm thanh ra các thiết bị output. |
| Output | Âm thanh có thể xuất ra thông qua audio listener hoặc audio mixer. |
| Mute | Nếu lựa chọn này bật , âm thanh sẽ phát nhưng bị mất tiếng. |
| Bypass Effects | Lựa chọn này giúp áp dụng nhanh các hiệu |

|  |  |
| --- | --- |
| **Thuộc tính** | **Ý nghĩa** |
|  | ứng cho nguồn âm thanh. Đây là một cách bật  ,tắt dễ dàng cho tất cả các hiệu ứng |
| Play On Awake | Nếu lựa chọn chức năng này , âm thanh sẽ bắt đầu phát ngay khi hoạt cảnh bắt đầu . Nếu không chọn chức năng này , thì âm thanh sẽ không tự động bật , bật âm thanh bằng cách sử dụng lệnh Play() |
| Loop | Làm cho tệp âm thanh phát lại |
| Priority | Xác định mức độ ưu tiên của nguồn âm thanh này đối với tất cả các nguồn âm thanh cùng tồn tại trong hoạt cảnh. (Mức độ ưu tiên: 0 = quan trọng nhất. 256 = ít quan trọng nhất. Mặc định  = 128.). Sử dụng 0 cho các bản nhạc để tránh bị thay đổi. |
| Volume | Tùy chỉnh mức độ to nhỏ của âm thanh |
| Pitch | Thay đổi tốc độ phát của nguồn âm thanh , 1 là tốc độ phát bình thường |
| Stereo Pan | Đặt vị trí trong trường âm thanh nổi của âm thanh 2D. |
| Doppler Level | Xác định có bao nhiêu hiệu ứng doppler sẽ được áp dụng cho nguồn âm thanh này (nếu được đặt thành 0, thì không có hiệu ứng nào được áp dụng). |
| Max Distance | Tùy chỉnh khoảng cách tối đa của âm thanh trong không gian 3D |

Bảng 2.2 Các chức năng tùy chỉnh Audio

Theo thông tin từ trang chủ chính thức của Unity thì các định dạng âm thanh được hỗ trợ được thể hiện trong bảng sau:



Hình 2.21 Danh mục các định dạng âm thanh được hỗ trợ

# Một số hàm thông dụng

- Xử lý và phát âm thanh với Script

Đầu tiên, để thao tác được với Audio Source, ta sử dụng dòng code sau:

m\_audio = GetComponent<AudioSource>();

Với C# Script, ta có thể thực hiện được mọi thao tác với AudioSource, nhưng tôi sẽ chỉ hướng dẫn các thao tác đơn giản và phổ biến.

-Thay đổi file âm thanh

File âm thanh sẽ được lưu trữ dưới biến clip. Do đó để thay đổi file âm thanh khác, ta gán file đó vào biến này. Thao tác như sau:

m\_audio.clip = yourAudioFile;

yourAudioFile phải được load lên bộ nhớ trước khi thực hiện dòng code trên.

-Play - Pause - Stop

Ba hàm này dùng để điều khiển bật / tắt nhạc theo nhu cầu của lập trình viên. Các hàm này sẽ thay đổi giá trị của biến isPlaying để thể hiện được trạng thái của AudioSouce.

* m\_audio.Play();
* m\_audio.Pause();
* m\_audio.Stop();

Hàm Pause sẽ dừng chơi nhạc và lưu trữ trạng thái hiện tại để có thể resume, trong khi hàm Stop sẽ dừng hoàn toàn và di chuyển về vị trí đầu của file âm thanh trong bộ nhớ.

-isPlaying

Biến này có tác dụng như một cờ đánh dấu để AudioSource có thể kiểm tra và bật/tắt âm thanh theo ý lập trình viên. Để kiểm tra giá trị của biến, ta có thể sử dụng hàm Debug.Log như sau:

Debug.Log(m\_audio.isPlaying);

* 1. COROUTINE

Khi ta thực thi một hàm, hàm đó sẽ được xử lý hoàn toàn trước khi trả về một giá trị. Điều đó có nghĩa là dù hàm có phức tạp đến mấy cũng sẽ được xử lý trong duy nhất một vòng lặp của chương trình. Do đó, các hàm dạng này không thể được sử dụng để xử lý các hiệu ứng, animation hoặc các sự kiện diễn ra trong nhiều thời gian.

Coroutine là một thành phần của chương trình máy tính giúp chúng ta tạo ra các tiến trình con độc lập, bằng cách cho phép tại một vị trí có thể có nhiều entry point hoạt động trong cùng thời điểm. Trong Unity, Coroutine được sử dụng để bắt đầu một tiến trình độc lập với hàm Update và được gọi một lần duy nhất. Unity sẽ tự động quản lý tiến trình này cho đến khi nó kết thúc và được giải phóng.

Dưới đây là 1 số các từ khóa và phương thức cần chú ý khi sử dụng Coroutine

# Từ khoá yield

Từ khoá yield được ứng dụng nhiều trong các ngôn ngữ lập trình. yield là một tín hiệu để báo hiệu cho trình biên dịch biết đoạn code hay hàm chứa nó chính là một khối lặp, mặc dù không sử dụng các cú pháp thông thường đã biết. yield sẽ kết hợp với return, cho phép trả về các giá trị của khối lặp đó, đồng thời có thể quay trở lại khối lặp trong những lần lặp tiếp theo.

yield return sẽ không kết thúc phương thức chứa nó mà vẫn tiếp tục chạy cho đến khi thực thi xong lệnh cuối cùng của khối lặp. Muốn kết thúc phương thức, ta sử dụng yield break. Phương thức chứa yield có kiểu trả về là IEnumerator, được sử dụng nhằm tạo ra một Coroutine.

# Phương thức WaitForSeconds

WaitForSeconds là một phương thức của Unity có chức năng trì hoãn một Coroutine với một khoảng thời gian tính bằng giây. WaitForSeconds được sử dụng duy nhất với từ khoá yield

# Sử dụng Coroutine trong Unity

Ta dùng phương thức StartCoroutine để bắt đầu một coroutine trong Unity. Nguyên mẫu của hàm như sau:

public Coroutine StartCoroutine(IEnumerator routine);

Tham số nhận vào là một IEnumerator, giá trị trả về của một hàm IEnumerator. Trong Unity, có thể có nhiều Coroutine hoạt động đồng thời trong cùng một thời gian. Ngoài ra, ta cũng có thể truyền trực tiếp tên hàm IEnumerator vào hàm dưới dạng chuỗi.

Ví dụ sử dụng Coroutine trong Unity:

void Start()

{

//in ra tab console thời gian bắt đầu Debug.Log("Starting " + Time.time);

//bắt đầu sử dụng coroutine StartCoroutine(PrintfAfter(2.0f));

}

IEnumerator PrintfAfter(float seconds)

{

//hàm thực hiện đợi trong khoảng thời gian là giá trị của biến seconds

yield return new WaitForSeconds(seconds);

//in ra tab console thời gian sau khi thực hiện xong

Debug.Log("Done " + Time.time);

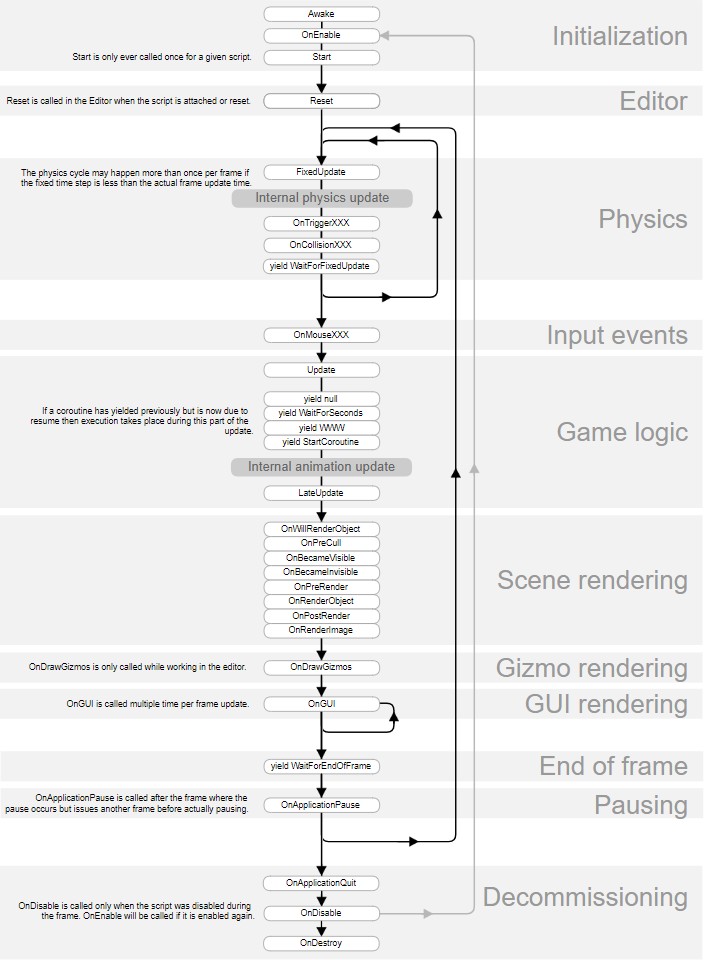
}

Hàm PrintfAfter có kiểu trả về là IEnumerator. Dòng số 9, cặp từ khoá

yield return, kết hợp với WaitForSeconds sẽ dừng hàm và bỏ qua dòng lệnh số 10 cho đến khi khoảng thời gian đã trôi qua hết. Khi đó dòng lệnh số 10 mới được thực thi và hàm sẽ thật sự kết thúc và được giải phóng.

* 1. CÁC HÀM SỰ KIỆN VÀ THỨ TỰ HOẠT ĐỘNG

Trong tập lệnh Unity, có một số hàm sự kiện được thực hiện theo thứ tự được xác định trước khi một tập lệnh được thực thi. Các hàm sự kiện này được thực thi thứ tự theo mô tả hình ảnh dưới đây:



Hình 2.22 Các hàm sự kiện và thứ tự hoạt động

* Hàm Awake() : Hàm này luôn được gọi trước mọi hàm Start và cũng được gọi sau khi một Game Object được khởi tạo.
* Hàm OnEnable(): (chỉ được gọi nếu Object đang hoạt động): Hàm này được gọi ngay sau khi đối tượng được kích hoạt. Điều này xảy ra khi một đối tượng thừa kế lớp MonoBehaviour được tạo ra
* Hàm OnLevelWasLoaded() : Hàm này được thực hiện để thông báo cho trò chơi rằng một level mới đã được load.
* Hàm Reset() :Hàm này được gọi để khởi tạo thuộc tính của tập lệnh khi

nó được đính kèm lần đầu tiên vào đối tượng và cũng có thể khi lệnh Reset được sử dụng.

* Hàm Start() :Hàm Start được gọi trước khi cập nhật khung đầu tiên chỉ

khi 1 đối tượng chứa script được bật.

* Đối với các đối tượng được thêm vào cảnh, hàm Start sẽ được gọi trên tất cả các tập lệnh trước hàm Update. Việc này không thể thực thi khi một đối tượng được khởi tạo trong khi ta đang chạy game.
* Hàm OnApplicationPause() : Hàm này được gọi ở cuối cùng mỗi khung hình. Một khung phụ sẽ tạo sau khi hàm OnApplicationPause được gọi để cho phép trò chơi hiển thị đồ họa cho trạng thái tạm dừng.
* Hàm FixedUpdate(): Hàm FixedUpdate thường được gọi thường xuyên hơn hàm Update. Nó có thể được gọi nhiều lần trên mỗi khung hình nếu tốc độ khung hình thấp và nó có thể không được gọi giữa các khung hình nếu tốc độ khung hình cao. Tất cả các tính toán vật lý diễn ra ngay lập tức sau hàm FixedUpdate. Khi áp dụng các phép tính chuyển động bên trong FixedUpdate, ta không cần nhân các giá trị của mình với Time.deltaTime. Điều này là do FixedUpdate được gọi trên một bộ đếm thời gian chuẩn, độc lập với tốc độ khung hình.
* Hàm Update() : Hàm Update được gọi một lần cho mỗi khung hình.
* Hàm LateUpdate(): Hàm LateUpdate được gọi tại mỗi khung hình, ngay sau khi hàm Update thực thi xong. Bất kỳ tính toán nào được thực hiện trong hàm Update sẽ hoàn thành khi LateUpdate bắt đầu. Hàm LateUpdate được sử dụng phổ biến với những game có camera theo góc nhìn thứ ba. Nếu ta làm cho nhân vật di chuyển trong hàm Update, ta có thể thực hiện tất cả các tính năng di chuyển và xoay của camera trong hàm LateUpdate. Điều này sẽ đảm bảo rằng nhân vật của ta sẽ di chuyển hoàn toàn trước khi camera theo dõi vị trí của nhân vật đó.
* Hàm OnPreCull(): Hàm này được gọi trước khi Camera lựa chọn hoạt cảnh.Việc lựa chọn xác định đối tượng nào sẽ được hiển thị trong camera. Hàm OnPreCull được gọi ngay trước khi việc lựa chọn diễn ra.
* Hàm OnBecameVisible / OnBecameInvisible() : Được gọi khi một đối

tượng hiển thị / ẩn đi với bất kỳ Camera nào.

* Hàm OnWillRenderObject: Được gọi một lần cho mỗi Camera nếu đối tượng hiển thị.
* Hàm OnPreRender() : Được gọi trước khi Camera bắt đầu hiển thị hoạt cảnh.
* Hàm OnRenderObject() : Được gọi sau khi tất cả hiển thị cảnh thường

được thực hiện. Chúng ta có thể sử dụng lớp GL hoặc Graphics.DrawMeshNow để vẽ hình dạng tùy chỉnh tại thời điểm này.

* Hàm OnPostRender() : Được gọi sau khi máy ảnh kết thúc hiển thị

cảnh.

* Hàm OnRenderImage() : Được gọi sau khi hiển thị cảnh hoàn tất để cho phép xử lý sau hình ảnh, xem Hiệu ứng sau xử lý.
* Hàm OnGUI() : Được gọi nhiều lần trên mỗi khung hình để phản hồi các sự kiện GUI. Sự kiện bố trí và vẽ lại được xử lý trước tiên, theo sau là sự kiện Bố cục và bàn phím / chuột cho mỗi sự kiện đầu vào.
* Hàm OnDrawGizmos() : Được sử dụng để vẽ Gizmos trong khung cảnh cho mục đích hiển thị.
* Hàm OnDestroy() : Hàm này được gọi sau khi tất cả các hàm sự kiện

khác hoạt động xong . Hàm này được sử dụng để xóa bỏ 1 đối tượng.

* Hàm OnApplicationQuit(): Hàm này được gọi trên tất cả các đối tượng trước khi ứng dụng thoát. Trong trình chỉnh sửa, nó được gọi khi người dùng dừng game.
* Hàm OnDisable(): Hàm này được gọi khi đối tượng game trở nên vô hiệu hóa hoặc không hoạt động.