TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

TẤT TUẤN PHONG – HOÀNG PHƯƠNG

PHÁT TRIỂN GAME 3D VỚI UNITY TRÊN MÔI TRƯỜNG ANDROID

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN CNTT

TP.HCM, 2011

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

TẤT TUẤN PHONG – 0841140
HOÀNG PHƯƠNG – 0841144

PHÁT TRIỂN GAME 3D VỚI UNITY TRÊN MÔI TRƯỜNG ANDROID

KHÓA LUẬN TỐT NGHIỆP CỬ NHÂN TIN HỌC

GIÁO VIÊN HƯỚNG DẪN
TS.TRẦN MINH TRIẾT – TRẦN DUY QUANG

NIÊN KHÓA 2008 – 2011

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

Khóa luận đáp ứng yêu cầu của LV cử nhân tin học.

TpHCM, ngày …… tháng …… năm 2011

Giáo viên hướng dẫn

NHẬN XÉT CỦA GIÁO VIÊN PHẢN BIỆN

……

……

……

……

……

……

……

……

……

……

……

……

……

……

……

Khóa luận đáp ứng yêu cầu của LV cử nhân tin học.

TpHCM, ngày …… tháng …… năm 2011

Giáo viên phản biện

LỜI CÁM ƠN

Chúng em xin chân thành cảm ơn Khoa Công Nghệ Thông Tin, trường Đại Học Khoa Học Tự Nhiên, Tp.HCM đã tạo điều kiện tốt cho chúng em thực hiện đề tài này.

Chúng em xin chân thành cảm ơn Thầy Trần Minh Triết, là người đã tận tình hướng dẫn, chỉ bảo chúng em trong suốt thời gian thực hiện đề tài. Chúng em cũng xin cảm ơn bạn Võ Quang Việt đã có những trao đổi, những chỉ dẫn giúp chúng em giải quyết các vấn đề và hoàn thiện đề tài.

Chúng em cũng xin gửi lời cảm ơn sâu sắc đến quý Thầy Cô trong Khoa đã tận tình giảng dạy, trang bị cho chúng em những kiến thức quí báu trong những năm học vừa qua.

Chúng em xin gửi lòng biết ơn sâu sắc đến Ba, Mẹ, các anh chị và bạn bè đã ủng hộ, giúp đỡ và động viên chúng em trong những lúc khó khăn cũng như trong suốt thời gian học tập và nghiên cứu.

Mặc dù chúng em đã cố gắng hoàn thành luận văn trong phạm vi và khả năng cho phép, nhưng chắc chắn sẽ không tránh khỏi những thiếu sót, kính mong sự cảm thông và tận tình chỉ bảo của quý Thầy Cô và các bạn.

Nhóm thực hiện

Tất Tuấn Phong & Hoàng Phương

ĐỀ CƯƠNG CHI TIẾT

Tên Đề Tài: Phát triển game 3D với Unity trên môi trường Android

Giáo viên hướng dẫn: TS. Trần Minh Triết, Trần Duy Quang

Thời gian thực hiện: từ ngày 15/08 /2010 đến ngày 20/02/2011

Loại đề tài: Tìm hiểu công nghệ và xây dựng ứng dụng

Nội Dung Đề Tài (mô tả chi tiết nội dung đề tài, yêu cầu, phương pháp thực hiện, kết quả đạt được, …):

- Nghiên cứu tổng quan về Engine Unity
- Xác định, phân tích các vấn đề và đưa ra các giải pháp trong quá trình xây dựng game cho Android với Unity.
- Mô tả nội dung, đặc trưng của dòng game sẽ xây dựng
- Dùng Unity xây dựng thử nghiệm 1 game 3D cho Android thuộc dòng game chơi theo lượt.

Kế Hoạch Thực Hiện:

- 15/08/2010 - 15/09/2010: Tìm hiểu nền tảng lập trình Android.
- 16/09/2010 - 16/10/2010: Tìm hiểu cách lập trình game trên Android.
- 17/10/2010 - 17/11/2010: Tìm hiểu các thư viện làm game 3D trên Android.
- 18/11/2010 - 18/12/2010: Tìm hiểu các Game Engine hỗ trợ làm game 3D trên Android.
- 19/12/2010 - 19/1/2011: Tìm hiểu các phương pháp xây dựng game 3D trên Android với Unity.
- 20/01/2011 - 25/01/2011: Phân tích các vấn đề phát sinh khi xây dựng game 3D chơi theo lượt, đồng thời đưa ra các giải pháp.

• 26/01/2011 - 02/01/2011: Mô tả đặc trưng kiến trúc và nội dung game. • 03/01/2011 - 20/02/2011: Xây dựng game đã chọn.	
Xác nhận của GVHD	**Ngày 20 tháng 02 năm 2011** **Nhóm SV Thực hiện** **Tất Tuấn Phong – Hoàng Phương**

MỤC LỤC

DANH MỤC CÁC HÌNH

TÓM TẮT KHÓA LUẬN

Ngày nay, điện thoại di động đã trở thành phương tiện không thể thiếu trong cuộc sống hằng ngày của chúng ta. Điện thoại không còn đơn thuần là dành cho những cuộc gọi hay nhắn tin nữa, nhu cầu sử dụng điện thoại bây giờ rất đa dạng và phong phú, trong đó không thể bỏ qua nhu cầu giải trí. Game trên trên điện thoại di động đã trở thành thú vui số 1 của giới trẻ.

Tuy game trên điện thoại di động chơi không sướng như trên máy tính nhưng tính tiện lợi thì rất rõ, có thể chơi mọi lúc mọi nơi. Trước kia các ứng dụng game trên di động hầu hết là đồ họa 2D, nhưng với sự bùng nổ công nghệ hiện nay dẫn đến các thiết bị di động có đủ sức mạnh để chạy được các ứng dụng đòi hỏi cấu hình cao, các ứng dụng đồ họa phức tạp trong không gian 3 chiều.

Chính vì vậy mà nhu cầu chơi game 3D trên di động trở nên tăng cao, các hãng sản xuất liên tục cho ra đời những thể loại game 3D hình ảnh đẹp mắt, âm thanh sống động chạy trên các thiết bị di động. Song song đó, các game Engine 3D cũng không ngừng phát triển và hỗ trợ làm game cho các thiết bị di động với nhiều hệ điều hành khác nhau như Android, iOS.

Nội dung luận văn chúng em thực hiện bao gồm việc tìm hiểu Engine Unity và sử dụng Unity để xây dựng game 3D cho điện thoại di động trên môi trường Android.

Chương 1

Mở đầu

✍ *Nội dung Chương 1 trình bày sự phát triển của Android, sự ra đời các thiết bị di động dòng Android, giới thiệu một số game engine điển hình. Phần cuối sẽ trình bày về mục tiêu và ý nghĩa của đề tài.*

1.1. Giới thiệu chung

1.1.1. Sự phát triển của Android

Vào tháng 10/2008 hệ điều hành Android đã xuất hiện lần đầu tiên trên di động. Trong suốt năm 2009 thì điện thoại chạy hệ điều hành Android chỉ đếm được trên đầu ngón tay. Nhưng thật sự đáng kinh ngạc khi trong năm 2010 thì hàng trăm thiết bị sử dụng nền tảng Android đã xuất hiện rầm rộ, từ các hãng tên tuổi lớn cho đến các điện thoại vô danh ở Trung Quốc.

Các phiên bản mới liên tục được cập nhật, nhờ vậy mà các nhà phát triển tự do xây dựng các ứng dụng trên hệ điều hành mở là điểm làm cho Android phát triển nhanh chóng. Lôi kéo đầy đủ các tên tuổi lớn như Samsung, LG, Sony Ericsson, HTC, Motorola... trong đó, tất cả đều coi Android là nền tảng chính cho thấy Google rất khôn khéo. Và cũng chính Android đã giúp Motorola thoát khỏi giai đoạn khủng hoảng.

Android đã trở thành hệ điều hành hàng đầu và rất phổ biến trên thế giới hiện nay. Rất nhiều hãng sản xuất đã cho ra đời nhiều mẫu máy cũng như chủng loại sử dụng hệ điều hành Android từ phổ thông cho đến siêu cấp.

(a) *T-Mobile G1 (HTC Dream)*

(Nguồn: http://www.htc.com)

(b) Motorola-DROID

(Nguồn:http://www.htc.com/)

(c) Orange San Francisco Android

(Nguồn: http://www.slashphone.com/)

(d) Samsung Galaxy S

(Nguồn: http://www.samsung.com/vn)

(e) Sony Ericsson Xperia™ X1

(Nguồn: http://www.sonyericsson.com/)

(f) Galaxy Tab dùng Android

(Nguồn: http://www.sonyericsson.com/)

Hình 1.1 Một số hình ảnh điện thoại chạy hệ điều hành Android

Canalys, một công ty phân tích thị trường lớn tại Mỹ mới đây đã đưa ra những báo cáo chứng minh hệ điều hành Android đã vượt xa Symbian trên thị trường điện thoại di động thông minh:

Worldwide smart phone market
Market shares Q4 2010, Q4 2009

OS vendor	Q4 2010 shipments (millions)	% share	Q4 2009 shipments (millions)	% share	Growth Q4'10/Q4'09
Total	101.2	100.0%	53.7	100.0%	88.6%
Google*	33.3	32.9%	4.7	8.7%	615.1%
Nokia	31.0	30.6%	23.9	44.4%	30.0%
Apple	16.2	16.0%	8.7	16.3%	85.9%
RIM	14.6	14.4%	10.7	20.0%	36.0%
Microsoft	3.1	3.1%	3.9	7.2%	-20.3%
Others	3.0	2.9%	1.8	3.4%	64.8%

*Note: The Google numbers in this table relate to Android, as well as the OMS and Tapas platform variants
Source: Canalys estimates, © Canalys 2011

Hình 1.2 Bảng thống kê tình hình tiêu thụ điện thoại của các hãng

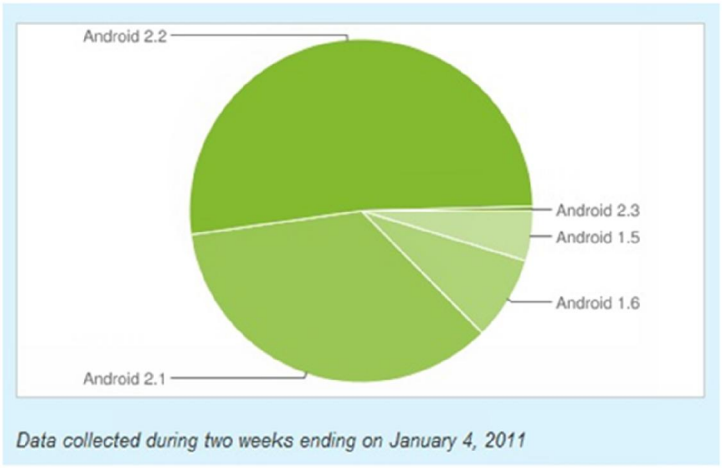
(Nguồn: http://www.canalys.com/)

Những chiếc điện thoại thông minh sử dụng hệ điều hành Android đã vượt qua mặt Nokia và Apple để dẫn đầu thị trường điện thoại thông minh thế giới. Dựa theo các số liệu phân tích thị trường của năm 2009 và năm 2010, công ty chuyên phân tích thi trường Canalys đã đưa ra báo cáo tình hình tăng trưởng của những hệ điều hành dành cho điện thoại trong quý IV năm 2010.

Báo cáo trên cho thấy, trong năm vừa qua, ở Quý IV, những chiếc điện thoại sử dụng nền tảng Android được người mua nhiều hơn so với số lượng bán ra của các dòng máy Symbian, với 33.300.000 chiếc điện thoại Android được xuất xưởng. Bên cạnh đó, hệ điều hành Android đã chiếm tới 32,6% người dùng trên toàn thế giới, nhiều hơn 2,9% so với hệ điều hành Symbian.

Báo cáo trên cũng chỉ rõ nhiều hãng sản xuất điện thoại đã tăng thị phần điện thoại thông minh nhanh chóng như LG, Samsung, Acer và HTC với mức tăng trưởng lần lượt là 4.127%, 1.474%, 709% and 371% so với cùng kỳ năm trước. Trong đó 2 hãng là HTC và Samsung đã chiếm gần 45% thị trường điện thoại sử dụng hệ điều hành Android.

Đây là một sự kiện quan trọng của thị trường di động bởi hơn 10 năm trở lại đây, chưa hề có bất kỳ hệ điều hành nào có thể vượt mặt được hệ điều hành Symbian của Nokia. Điều đáng nói, tốc độ tăng trưởng của Android diễn ra rất nhanh và vẫn chưa hề có dấu hiệu giảm lại.



Platform	API Level	Distribution
Android 1.5	3	4.7%
Android 1.6	4	7.9%
Android 2.1	7	35.2%
Android 2.2	8	51.8%
Android 2.3	9	0.4%

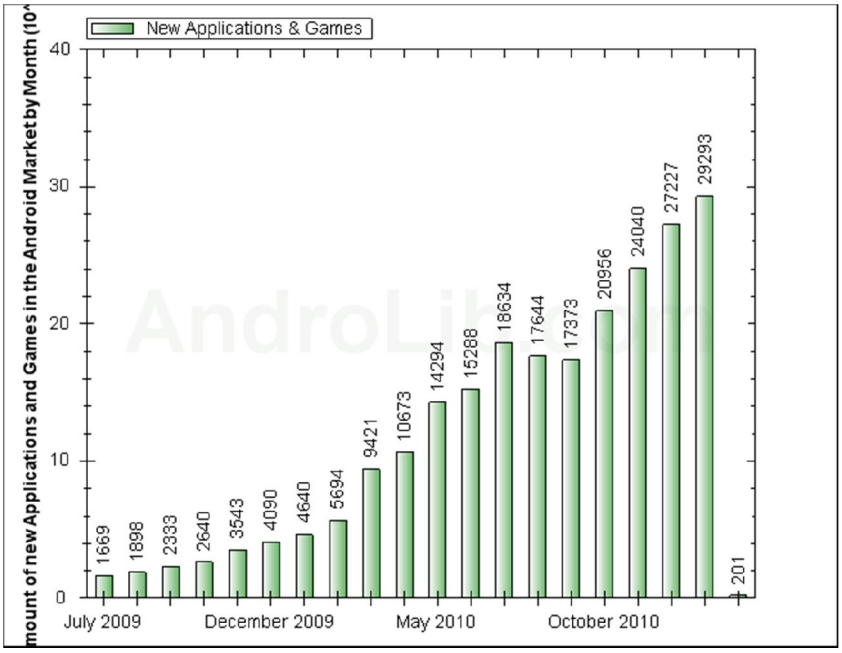
Hình 1.3 Thống kê tình hình các phiên bản Android

(Nguồn: http://developer.android.com/)

Theo bản phân tích đánh giá tình hình các phiên bản hệ điều hành Android tính đến giữa tháng 1/2011 cho thấy, phiên bản mới nhất của hệ điều hành này là Android 2.3 đã có mặt trong bản phân tích với 0,4%.

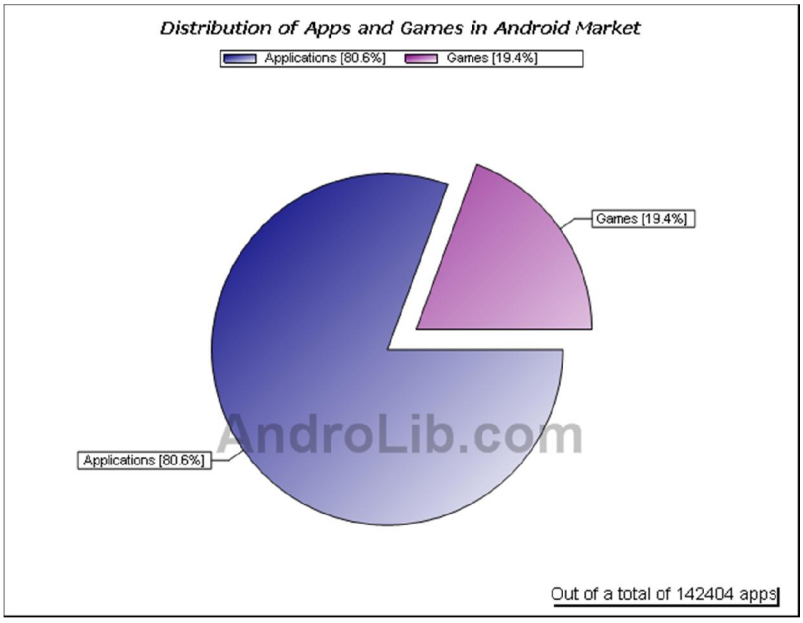
Như vậy là tính đến đầu năm 2011, tất cả phiên bản của hệ điều hành Android 2.x đã chiếm đến 87,4%, tăng thêm 4,4% so với đầu tháng 12/2010. Theo dự đoán trong tháng đầu năm mới này, hệ điều hành Android 2.1 sẽ có mức giảm nửa trong khi cả 2 phiên bản Android 2.2 và 2.3 sẽ có thêm mức tăng nhưng sẽ không cao như báo cáo của tháng này.

Thống kê mới nhất thì số lượng ứng dụng và game trên Android Market đã đạt tới con số 29.293 và game chiếm 19.4%.



Hình 1.4 Số liệu thống kê các ứng dụng mới được xuất bản theo tháng (bao gồm ứng dụng game)

(Nguồn: http://www.androlib.com)



Hình 1.5 Số liệu thống kê tỷ lệ giữa ứng dụng và game

(Nguồn: http://www.androlib.com/appstatstype.aspx)

1.1.2. Game 3D trên Android

Thị trường game đã sôi nổi từ nhiều năm nay. Trong thời gian gần đây, với sự hỗ trợ của những công nghệ tiên tiến, thế giới game 3D đã thật hơn và diễn tả được khá đầy đủ những gì tồn tại của cuộc sống thật.

Việc ứng dụng các cấu hình phần cứng dựng sẵn là xu hướng chung khi phát triển game mobile. Từ ứng dụng màn cảm ứng đa điểm cho tới hệ cảm biến hành vi giúp xoay ngang, xoay dọc để điều khiển nhân vật trong game đã trở thành mặc định trong các tựa game 3D cao cấp.

Năm 2010 kết thúc đánh dấu một nấc thang vượt bậc của các tựa game di động cùng sự bùng nổ của các nền tảng tiên tiến. Và chắc chắn, trong năm 2011 này hứa hẹn sẽ là thời khắc cao trào của sự cạnh tranh, phát triển và người dùng sẽ được chứng kiến một thời kỳ sôi động chưa từng thấy của những tựa game, cỗ máy di động đỉnh cao.

Một vài game 3D tiêu biểu trên Android:

(a) Game Hero of Sparta

(Nguồn: http://www.vietgiaitri.com/)

(b) Game đua xe Asphalt5

(Nguồn: http://tinhte.vn/)

(c) Real Football 2011 v3.1.2

(Nguồn: http://www.dalats.com/forum/)

(d) Brother in Arm 2

(Nguồn: http://www.eurodroid.com/)

Hình 1.6 Một số hình ảnh về game 3D trên Android

Hiện nay, có rất nhiều engine hỗ trợ làm game 3D trên Android. Do đó chúng em cần tìm được một game engine đủ tốt để có thể xây dựng game. Tiêu chí khảo sát của chúng em đó là engine đó phải render nhanh và nhiều mô hình cùng lúc trên màn hình, có thể làm mô hình nhân vật chuyển động, hỗ trợ va chạm giữa các vật thể, hỗ trợ hiệu ứng particle tốt. Sau đây là một số game engine 3D cho Android:

- **jPCT-AE:** jPCT-AE là một bản port từ engine jPCT sang cho Android. Đây là một engine 3D miễn phí, nhỏ gọn, tính năng tương đối ít.
- **libGDX:** là thư viện được viết chủ yếu bằng NDK (công cụ cho phép gọi thực thi code C/C++ từ Java) nên cho tốc độ xử lý nhanh. Một ưu điểm của engine này là nó cho phép chúng ta viết game và test hoàn toàn ngay trên nền desktop. Tuy nhiên,

lidGDX được phát triển lúc đầu chủ yếu dành cho nền 2D nên cho đến thời điểm này thư viện này vẫn chưa hỗ trợ kiểm tra va chạm giữa các vật thể trên nền 3D.

❖ **Shiva3D:** là một engine thương mại khá mạnh. Shiva dùng ngôn ngữ Lua để viết script trong game, một ngôn ngữ ít người biết đến.

❖ **Unity:** là một trong những game engine khá phổ biến hiện nay, có khả năng phát triển trò chơi đa nền, trình biên tập có thể chạy trên Windows và Mac OS, và có thể xuất ra game cho Windows, Mac, iOS, Android, Wii, Web, Xbox 360, PlayStation 3. Unity tạo ra được nhiều loại game 3D đa dạng, hỗ trợ import rất nhiều mô hình định dạng khác nhau, hỗ trợ tạo mô hình trực tiếp. Lượng tài liệu hướng dẫn nhiều, cộng đồng lớn với diễn đàn riêng. Unity có 2 phiên bản là Unity Pro có tính phí và Unity Free để người dùng dễ dàng lựa chọn, vì vậy mà Unity không chỉ dành cho một công ty lớn chuyên ngiệp, mà kể cả giới làm game không chuyên cũng có thể sử dụng được một cách dễ dàng.

Bởi các tính năng tuyệt vời và phổ biến của Unity, nên trong luận văn này nhóm chúng em được giao nhiệm vụ tìm hiểu.

1.2. Mục tiêu đề tài

Đề tài này thuộc hướng tìm hiểu công nghệ từ đó xây dựng ứng dụng. Mục tiêu của đề tài là tìm hiểu engine Unity và sử dụng Unity xây dựng thử nghiệm game 3D thể loại chơi theo lượt (turn-base) chạy trên môi trường Android cho thiết bị di động.

Để thực hiện được được điều này nội dung của luận văn bao gồm:

- Giới thiệu tổng quan về Android.
- Tìm hiểu tổng quan về kiến trúc của Unity và cách tạo lập các ứng dụng trong Unity.
- Tìm hiểu các vấn đề như load mô hình vào game, làm nhân vật chuyển động, cách tạo địa hình, giao diện, âm thanh và các hiệu ứng particle… để rồi từ đó đưa ra giải pháp.
- Xây dựng và phát triển ứng dụng game thể loại chơi theo lượt bằng Unity.

1.3. Nội dung luận văn

Luận văn bao gồm 5 chương:

Chương 1: Giới thiệu tổng quan về sự phát triển của Android các thiết bị di động trên môi trường Android, game 3D trên Android.

Chương 2: Giới thiệu tổng quan về Unity, trình bày các khái niệm cơ bản, các tính năng nổi bật, giao diện, đặc biệt là tổng quan về kiến trúc của Engine Unity do chúng em xác định.

Chương 3: Trình bày các vấn đề và giải pháp khi xây dựng ứng dụng game 3D trên Unity cho Android.

Chương 4: Trình bày một số đặc trưng chính của ứng dụng game xây dựng và kiến trúc trong game.

Chương 5: Kết luận và hướng phát triển.

Chương 2

Tổng quan về engine Unity

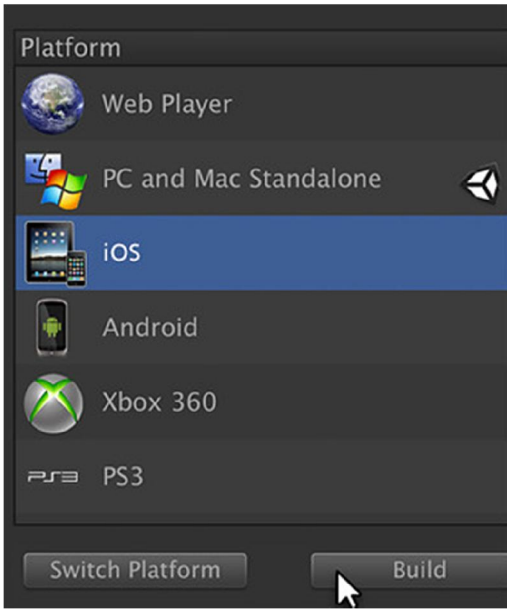
✍ *Nội dung chương này giới thiệu chung về engine Unity và nêu lên các khái niệm cơ bản trong Unity, đặc biệt trong chương này còn trình bày kiến trúc tổng quan của Unity mà nhóm đã xác định được từ những thành phần mà Unity cung cấp.*

2.1. Unity là gì?

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

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



Hình 2.1 Hình minh họa đa nền

(Nguồn: http://unity3d.com/)

Chỉ với khoản tiền bỏ ra khá khiêm tốn (1.500 USD) 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 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: nếu chúng ta là một người chuyên dùng 3Dmax, hay Maya hoặc phần mềm mã nguồn mở Blender thì quả là thật tuyệt, chúng ta sẽ có một lợi thế lớn khi viết game trên Unity này, bởi công việc tạo các mô hình 3D sẽ trở lên dễ dàng hơn rất nhiều, việc kết hợp giữa người lập trình và người thiết kế các mô hình sẽ nhanh và hiệu quả hơn. Trong Unity chúng ta có thể import trực tiếp các file mô hình đang thiết kế và sẽ thiết kế hoàn thiện tiếp nếu chưa xong trong khi đó công việc import chỉ diễn ra một lần. Không như việc phải dùng các công cụ khác để thực hiện viết game chúng ta sẽ phải xuất chúng ra một dạng nào đó và mỗi lần sửa lại phần mô hình chúng ta lại phải import lại, và như thế là quá mất thời gian trong việc tạo và chỉnh sửa các mô hình theo ý muốn. Ngoài ra Unity còn cho chúng ta trực tiếp tạo các mô hình nếu muốn. Việc đặt các thuộc tính vật lý trong Unity cũng cực kỳ dễ dàng và hỗ trợ sẵn nhiều chức năng.

2.2. Sơ lược lịch sử hình thành và phát triển của Unity

2.2.1. 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. Có thể thấy tốc độ phát triển của Unity khá nhanh.

2.2.2. 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.

- 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

2.2.3. 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.

2.3. Tính năng của engine Unity

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

❖ Triển khai được trên nhiều nền tảng:

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

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

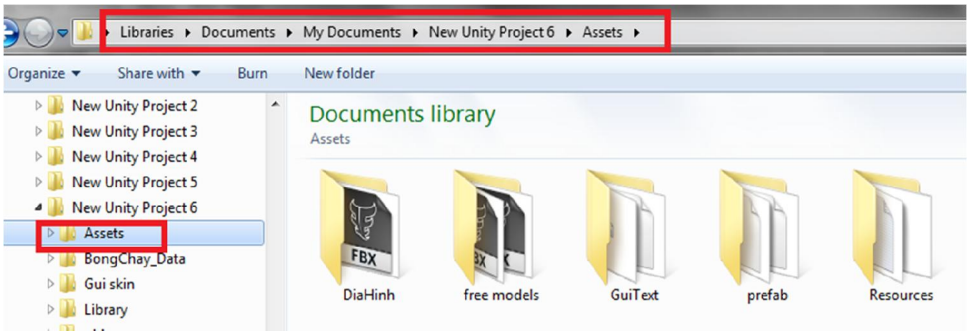
❖ Graphics engine sử dụng Direct3D (Windows), OpenGL (Mac, Windows), OpenGL ES (iPhone OS), và các API khác trên Wii.

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

2.4. Các khái niệm cơ bản trong Unity

2.4.1. Asset

Đây là kho tài nguyên cho việc xây dựng game trong một project của Unity. Các tài nguyên này có thể là hình ảnh, âm thanh, hoặc một mô hình 3D có sẵn. Unity sẽ tham chiếu đến các tập tin chúng ta sẽ sử dụng để tạo ra các tài nguyên cho trò chơi. Đây là lý do tại sao trong bất kỳ thư mục chứa project sử dụng Unity thì tất cả các tập tin tài nguyên phải được lưu trữ trong một thư mục con tên là Assets.

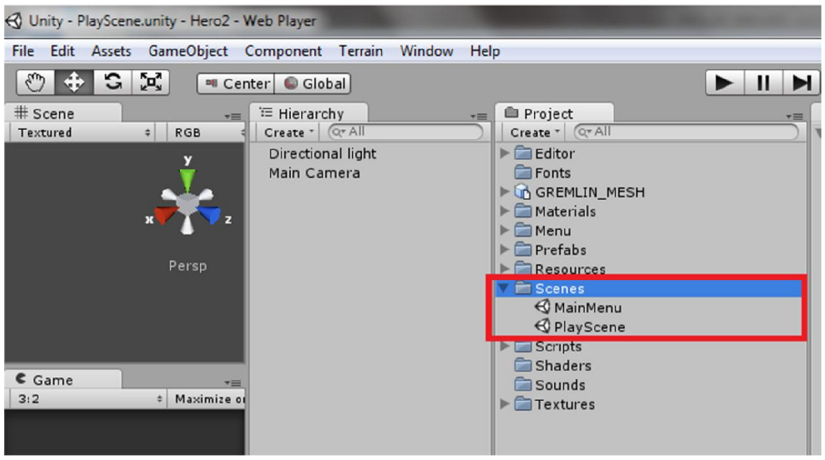


Hình 2.2 Asset trong Unity

2.4.2. Scene

Trong Unity, chúng ta có thể xem Scenes là các màn chơi, cấp độ chơi riêng lẻ, hoặc các vùng của nội dung trò chơi. Ví dụ như Main menu, Options, About …

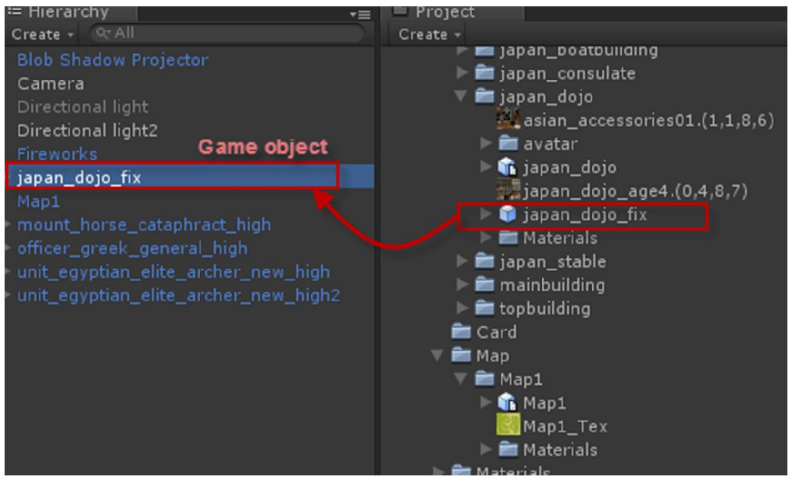
Bằng cách xây dựng trò chơi với nhiều cảnh, chúng ta sẽ có thể phân phối thời gian tải và thử nghiệm các phần khác nhau của trò chơi riêng lẻ một cách nhanh chóng và chính xác.



Hình 2.3 Các scene của Unity

2.4.3. Game Object

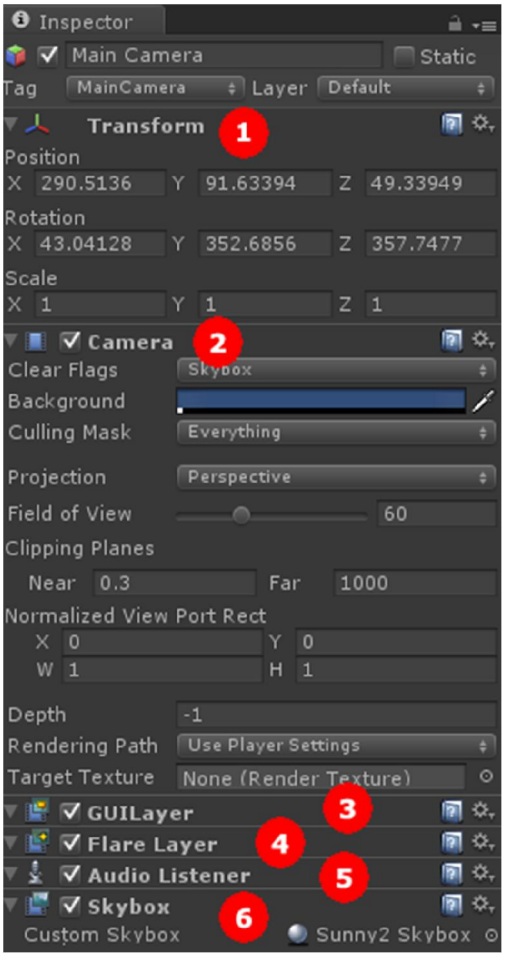
Khi một tài nguyên được sử dụng trong một scene, khi đó chúng ta có thể coi tài nguyên này là một "Game Object" mới. Mỗi GameObject phải chứa ít nhất một thành phần, đó là thành phần "Transform". Transform chứa các phép để biến đổi góc quay, tỷ lệ hay tịnh tiến của đối tượng. Từ đây trong báo cáo này chúng em sẽ gọi GameObject trong cửa sổ Hierarchy là *đối tượng game*.



Hình 2.4 Kéo tài nguyên vào Scene để sử dụng

2.4.4. Component

Component là các thành phần trong một Game Object của Unity. Bằng cách đính kèm các thành phần vào cho một đối tượng, chúng ta có thể áp dụng ngay các phần mới của game engine vào đối tượng. Thông thường các thành phần này được Unity xây dựng sẵn như ánh sáng, camera, particle, hiệu ứng vật lý…

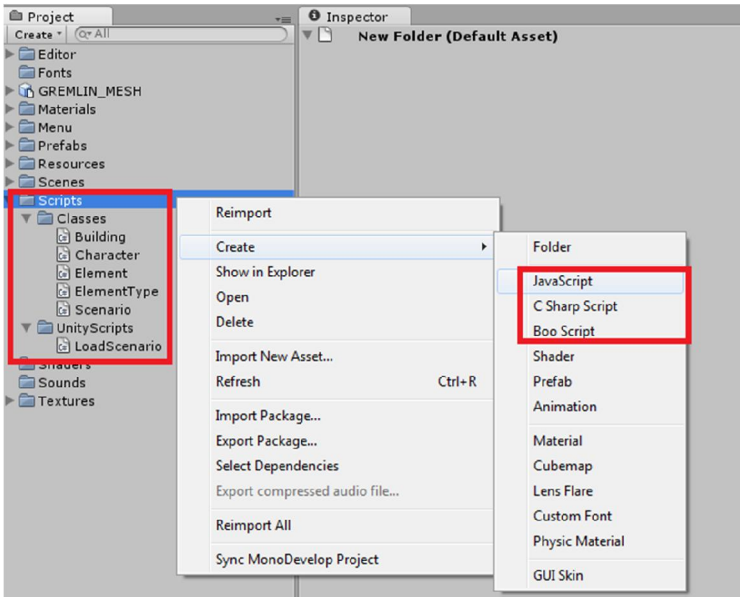


Hình 2.5 Các thành phần trong đối tượng Camera

2.4.5. Script

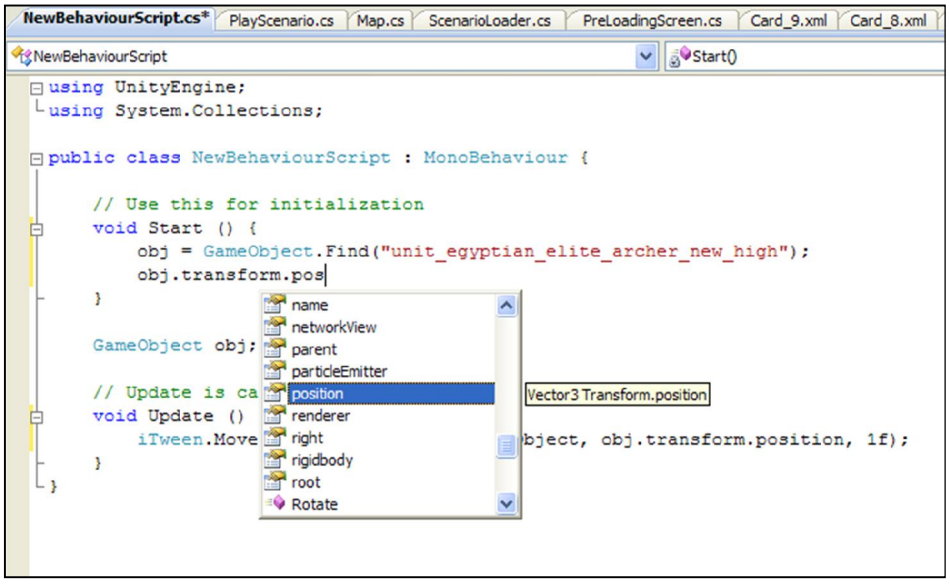
Script là thành phần quan trọng nhất trong Unity, có thể xem scripts như là linh hồn của game. Chúng ta có thể viết kịch bản cho game bằng C#, Java Scripts, hoặc Boo (một dẫn xuất của ngôn ngữ Python). Theo nhiều người đã sử dụng Unity thì code bằng C# sẽ giúp game chạy nhanh hơn và giúp kiểm soát code tốt hơn do tất cả các

biến phải được khai báo rõ ràng. Mặt khác ngôn ngữ C# rất tiện dụng để lập trình, nên trong luận văn này, chúng em dùng ngôn ngữ C# để viết kịch bản cho game. Mỗi file script C# là một class bắt buộc kế thừa từ lớp **MonoBehaviour**, có tên class phải trùng với tên file script.



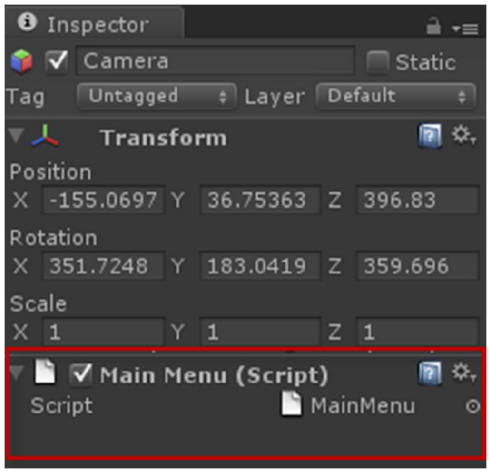
Hình 2.6 Cách tạo file script mới

Giao diện code C# rất quen thuộc khi edit trên Visual Studio 2005



Hình 2.7 Lập trình Unity bằng C# trên Visual Studio

Một đoạn script muốn thực thi được thì nó phải được gắn vào một đối tượng



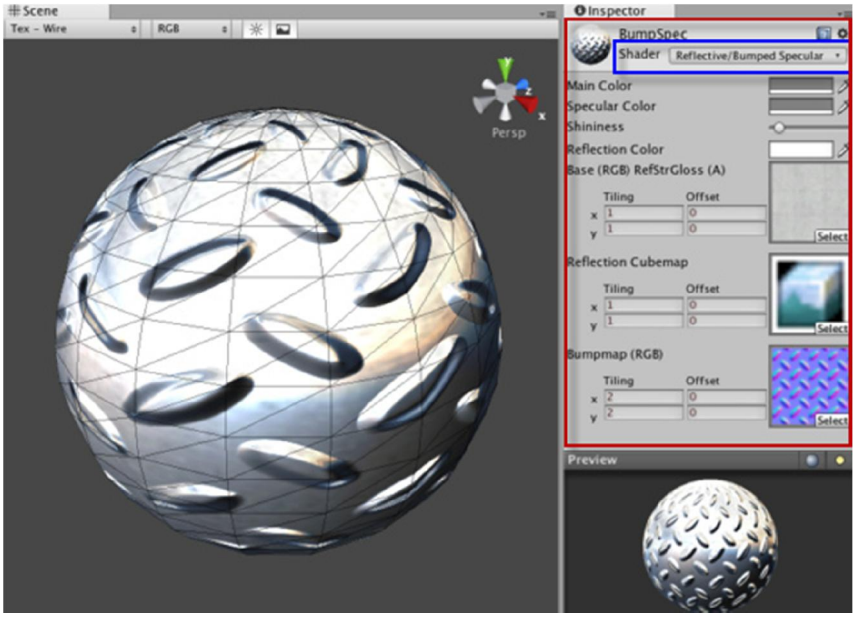
Hình 2.8 Một file script đang gắn vào đối tượng

2.4.6. Prefab

Hãy tưởng tượng Prefab là một cái thùng rỗng, mà bên trong nó chúng ta có thể chứa đựng các thành phần hay đối tượng khác nhau, chúng ta có thể viết kịch bản cho hành động của Prefab (khởi tạo, di chuyển, hay hủy đối tượng). Chúng ta có thể sử dụng đối tượng này nhiều lần trong trò chơi, và cũng có thể sử dụng lại cho project khác. Prefab cho phép chúng ta lưu trữ các đối tượng, toàn bộ thành phần bên trong và cấu hình hiện tại.

2.4.7. Material và Shader

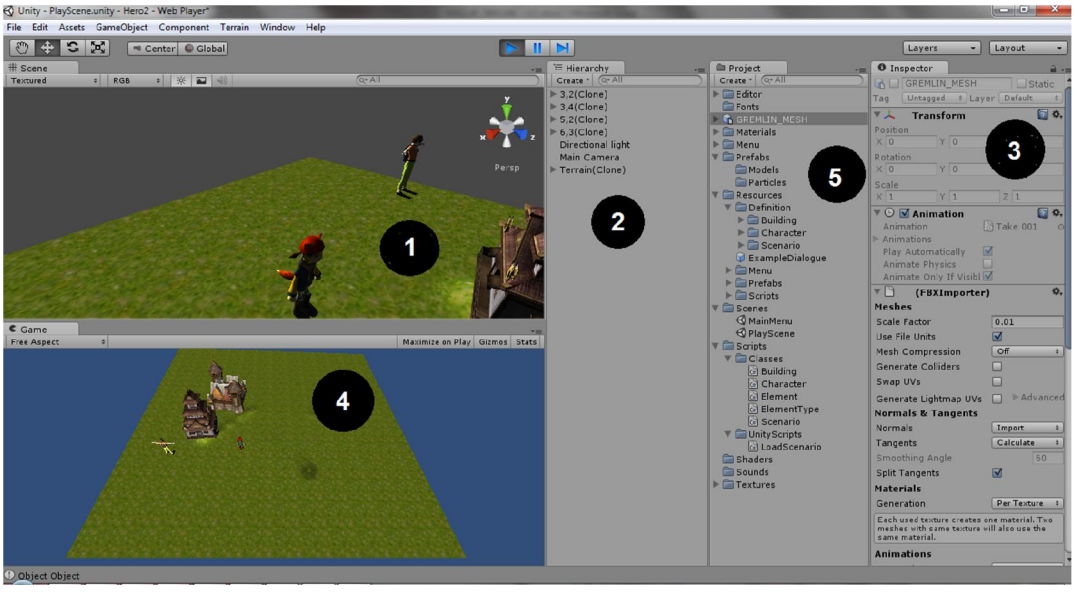
Shader là đoạn script qui định cách thức render của chất liệu trên bề mặt vật thể. Material sử dụng shader để làm chất liệu cho mô hình. Giữa materials và shaders có mối liên hệ với nhau. Shaders qui định các thuộc tính cần để shader làm việc. Còn material cho phép gán hình ảnh vào các thuộc tính đó từ Asset.



Hình 2.9 Material và Shader

2.5. Sơ nét về giao diện của Unity

Giao diện Unity, giống như nhiều môi trường làm việc khác, layout có thể tùy chỉnh. Layout của Unity bao gồm nhiều tab khác nhau và có thể bật tắt. Chúng ta hãy xem xét một cách bố trí giao diện Unity điển hình:



Hình 2.10 Giao diện Editor của Unity

Như hình trên chúng ta thấy có 5 khung khác nhau:

- Scene [1] – nơi xây dựng trò chơi
- Hierarchy [2] – danh sách các GameObject trong một cảnh game
- Inspector [3] – màn hình cài đặt cho tài nguyên/đối tượng đang được chọn
- Game [4] – cửa sổ xem trước game, chỉ hoạt động ở chế độ chơi (khi nhấn Play)
- Project [5] – danh sách các tài nguyên trong project, đóng vai trò như một thư viện.

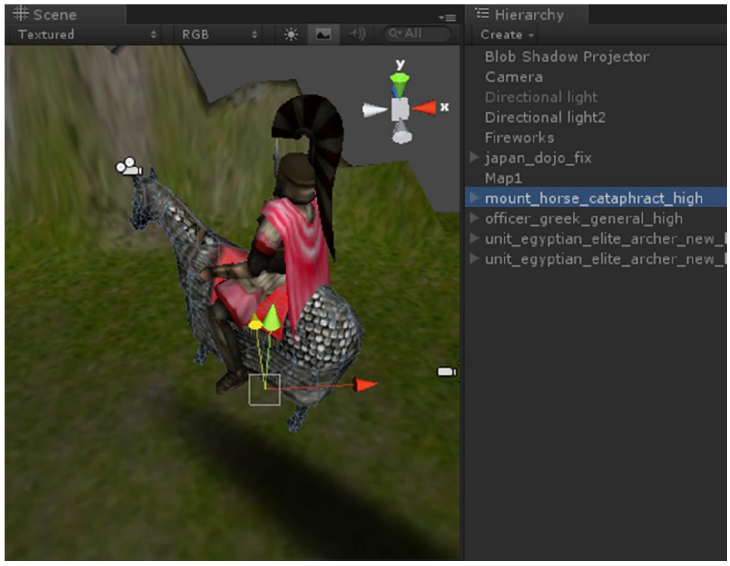
2.5.1. Cửa sổ Scene và Hierarchy

Cửa sổ Scene là nơi chúng ta sẽ xây dựng toàn bộ các đối tượng trong game. Cửa sổ cung cấp nhiều góc nhìn khác nhau, có thể nhìn dạng phối cảnh hoặc dạng song song. Chúng ta có thể kéo thả đối tượng trên cửa sổ này, di chuyển, xoay…

Cửa sổ Scene cũng kèm theo bốn nút điều khiển tiện lợi như hình trên. Truy cập từ bàn phím bằng cách sử dụng các phím Q, W, E, và R. Các phím thực hiện các hoạt động sau đây:

- Công cụ bàn tay [Q]: công cụ này cho phép di chuyển trong cửa sổ Scene, xoay góc nhìn, phóng to, thu nhỏ góc nhìn.
- Công cụ di chuyển [W]: Công cụ này dùng để di chuyển một đối tượng.
- Công cụ xoay [E]: Công cụ này cho phép chúng ta xoay nhân vật theo một trục nào đó trong không gian.
- Công cụ tỷ lệ [R]: Công cụ này cho phép chúng ta tăng giảm tỷ lệ kích thước của đối tượng.

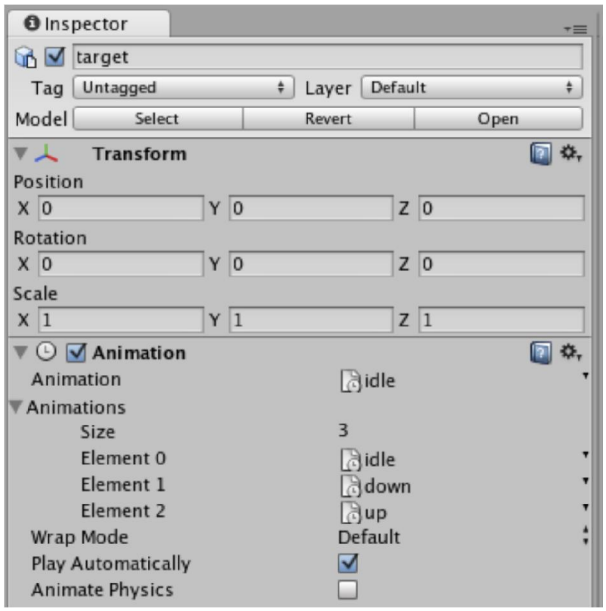
Khi chọn một đối tượng trong cửa sổ Scene, đối tượng này sẽ được tự động chọn trong cửa sổ Hierarchy và ngược lại.



Hình 2.11 Chọn đối tượng trong Scene và Hierrarchy

2.5.2. Inspector

Inspector sẽ hiển thị tất cả thông tin, các thành phần trong đối tượng game đang chọn, và cho phép điều chỉnh các biến của các thành phần này. Có thể xem cửa sổ này như cửa sổ Properties khi design giao diện Winform trên Visual Studio.



Hình 2.12 Cửa sổ Inspector hiển thị thông tin một đối tượng

2.5.3. Cửa sổ Game

Cửa sổ này sẽ hiển thị những gì có trong cửa sổ Scene và sẽ hoạt động khi nhấn nút Play. Trong cửa sổ này chúng ta có thể chọn các kích cỡ hiển thị khác nhau để build cho các loại máy khác nhau. Chúng ta có thể chơi thử game trên cửa sổ này khi đã nhấn nút Play. Lưu ý rằng khi cửa sổ này hoạt động rồi thì mọi chỉnh sửa trên cửa sổ Scene và cài đặt cho các đối tượng chỉ là tạm thời và khi nhấn nút Stop, cửa sổ này về lại trạng thái tĩnh thì mọi chỉnh sửa trước đó là không còn.

2.5.4. Cửa sổ Project

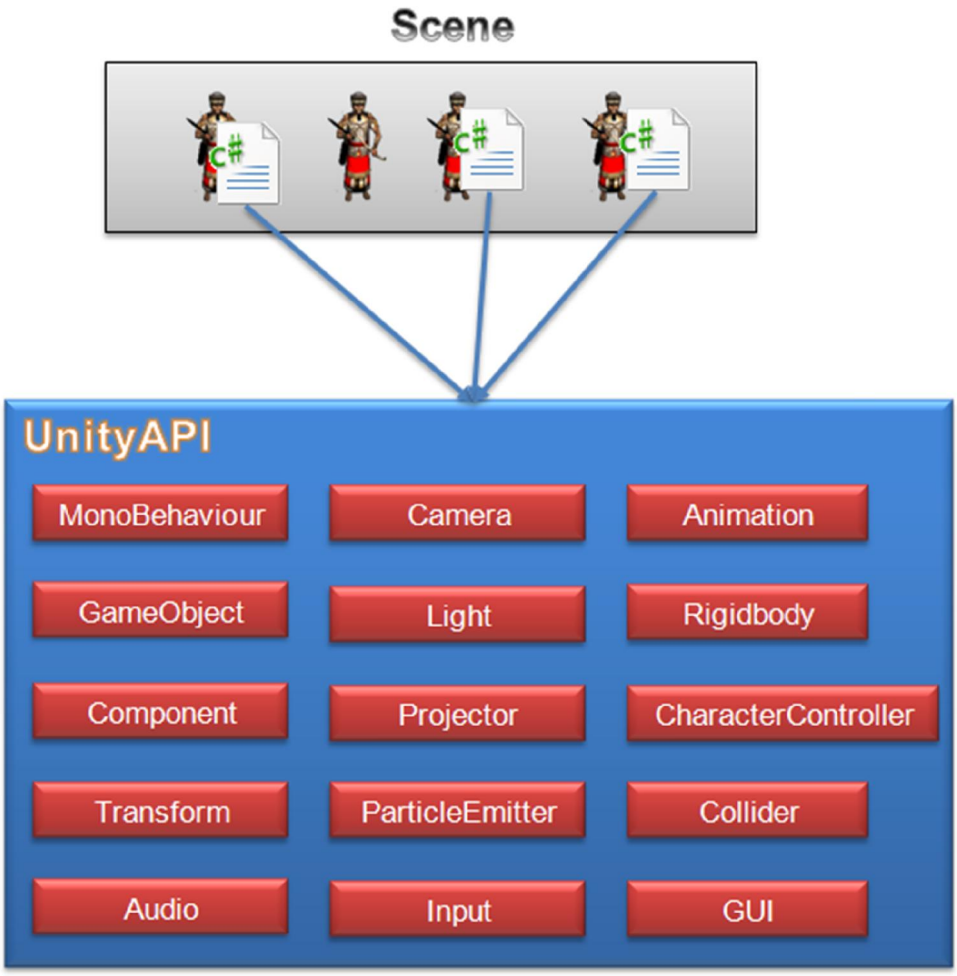
Cửa sổ Project thể hiện nội dung bên trong thư mục Assets của project chúng ta. Khi thêm tài nguyên vào thư mục Assets ngay lập tức chúng sẽ tự động được cập nhập vào project Unity của chúng ta.

2.6. Tổng quan kiến trúc engine Unity trên Android

2.6.1. Kiến trúc tổng quan

Engine Unity hỗ trợ cho chúng ta UnityAPI để viết script game. UnityAPI là API lập trình game trong Unity rất mạnh. UnityAPI chứa các đối tượng và phương thức hỗ trợ hầu hết các đối tượng và các loại thành phần trong Unity.

Trong một scene thường có nhiều đối tượng game. Mỗi đối tượng này có thể có hoặc không có đoạn script nào gắn lên đó. Nếu muốn gắn script vào đối tượng, ta bắt buộc phải kế thừa class đó từ lớp **MonoBehaviour** của UnityAPI và tên class phải trùng với tên file script. Mỗi script khi gắn lên đối tượng game đều được đối tượng game xem như một thành phần bên trong và được cấp phát vùng nhớ khi chạy game.



Hình 2.13 Tổng quan kiến trúc Unity

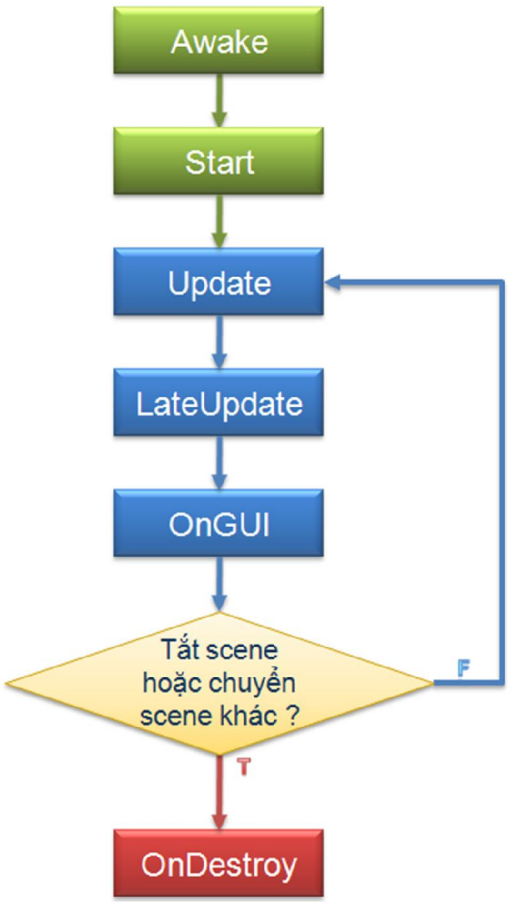
Bên trong UnityAPI chứa rất nhiều lớp hỗ trợ lập trình game, trong đó có một số lớp quan trọng như :

- **MonoBehaviour**: tất cả các script muốn gắn vào một đối tượng game bắt buộc phải kế thừa từ lớp này.
- **GameObject**: lớp cha của tất cả các thực thể trong scene.
- **Component**: lớp cha của tất cả các thành phần có thể gắn vào đối tượng.
- **Transform**: giúp thay đổi vị trí, xoay, biến đổi tỉ lệ mô hình.
- **Input**: hỗ trợ lập trình với chuột, cảm ứng đa điểm, cảm biến gia tốc.
- **Camera**: giúp lập trình camera.
- **Light**: giúp tạo ánh sáng trong game.
- **Projector**: giúp chiếu texture lên bề mặt vật thể.

- **ParticleEmitter**: hỗ trợ tạo các hiệu ứng particle đẹp mắt.
- **Audio**: hỗ trợ lập trình với âm thanh.
- **Animation**: chạy chuyển động của mô hình nhân vật.
- **Rigidbody**: giúp tạo hiệu ứng vật lý liên quan đến trọng lực như bóng nẩy, lăn, ..
- **CharacterController**: giúp điều khiển nhân vật di chuyển theo độ cao địa hình.
- **Collider**: hỗ trợ lập trình va chạm giữa các vật thể.
- **GUI**: giúp lập trình giao diện người dùng trên Unity.

2.6.2. Chu kỳ sống của thành phần script gắn trên đối tượng game

Chu kỳ sống của một thành phần script được tính kể từ khi scene được chạy cho đến lúc scene bị tắt hoặc chuyển sang scene khác.



Hình 2.14 Chu kỳ sống của thành phần script

Tùy theo trạng thái của scene mà sự kiện tương ứng sẽ được gọi.

- **Awake**: được gọi khi script được load xong
- **Start**: được gọi khi script được load xong nếu thành phần script đó không bị disable.
- **Update**: đây là sự kiện thường sử dụng nhất và được gọi liên tục từng frame để vẽ lại màn hình.
- **LateUpdate**: sự kiện này chạy sau sự kiện Update và được gọi liên tục từng frame.
- **OnGUI**: sự kiện này dùng để vẽ GUI và được gọi liên tục từng frame, chỉ trong sự kiện này ta mới có thể sử dụng các lớp hỗ trợ tạo giao diện người dùng của UnityAPI.
- **OnDestroy**: được gọi khi thành phần script bị hủy khỏi bộ nhớ.

2.7. Kết luận

Trong chương này chúng ta đã tìm hiểu tổng quan về Unity và nắm rõ các khái niệm cơ bản trong Unity. Chương tiếp theo ta sẽ tìm hiểu rõ hơn về cách lập trình game trên Unity.

Chương 3

Một số vấn đề và giải pháp khi xây dựng game với engine Unity trên Android

✍ *Nội dung chương này tập trung vào các vấn đề về xử lý mô hình, xây dựng địa hình, tạo các hiệu ứng trong game… và đồng thời đưa ra giải pháp xử lý cho từng vấn đề.*

3.1. Load mô hình 3D

Vấn đề

Game 3D được xây dựng từ nhiều mô hình 3D được đặt lên không gian 3 chiều sao cho hài hòa với nhau để tạo thành cảnh vật trong game. Do đó việc nạp và hiển thị được mô hình 3D trong game là vô cùng quan trọng.

Mô hình 3D được cấu tạo từ rất nhiều đa giác để tạo nên khối vật thể. Ngày nay, trong một mô hình 3D không chỉ đơn thuần chứa một khối vật thể mà nó bao gồm nhiều khối vật thể được gắn kết với nhau trên một khung xương. Điều này giúp cho mô hình không bị gắn chết một chuyển động vào bên trong và dễ dàng thay đổi chuyển động cho mô hình.

Giải pháp

Các mô hình 3D thông thường được thiết kế sẵn bằng các phần mềm thiết kế 3D chuyên dụng như 3DS Max, Blender, Cinema 4D, ... Sau đó, mô hình sẽ được đưa vào game engine để sử dụng.

Engine Unity hỗ trợ rất nhiều định dạng mô hình 3D khác nhau như: 3DS, OBJ, MAX, FBX, BLEND, MA, ... Tuy nhiên, khi import mô hình vào project thì Unity đều tự động chuyển mô hình đó sang định dạng FBX. Sau đó, mô hình sẽ được chuyển thành một Prefab để có thể tái sử dụng nhiều lần.

Unity có hỗ trợ load mô hình bằng cách kéo thả Prefab vào vị trí bất kỳ trong Scene. Tuy nhiên, để linh hoạt hơn thì chúng ta nên xử lý bằng code.

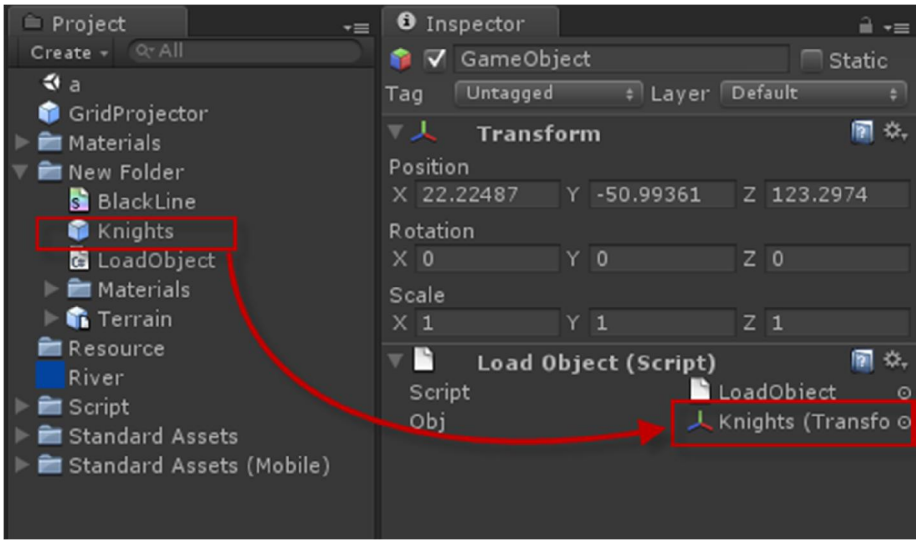
Trước tiên, ta tạo một file script và gắn nó vào một đối tượng game bất kỳ để đoạn script có thể thực thi. Trong file script này, ta khai báo một đối tượng kiểu **GameObject** để lưu mô hình và dùng hàm ***Instantiate()*** để khởi tạo mô hình này ở vị trí và góc quay mong muốn.

```
    public class LoadObject : MonoBehaviour
    {
        public GameObject Obj;

        void Awake()
        {
            GameObject NewObj = Instantiate(Obj, new Vector3(0, 0, 0),
Quaternion.identity);
        }
    }
```

Tuy nhiên, câu hỏi là đối tượng GameObject trên chứa mô hình nào. Ở đây có 2 giải pháp để trỏ GameObject vào mô hình.

- *Giải pháp thứ nhất:* Load mô hình từ prefab chứa bên ngoài resource

Trên cửa sổ Inspector của đối tượng game được gắn script vào xuất hiện thuộc tính Obj. Ta chọn Prefab mong muốn và kéo thả vào thuộc tính Obj.

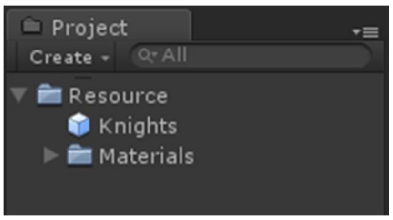


Hình 3.1 Minh họa kéo thả prefab vào thuộc tính của script

- *Giải pháp thứ hai:* Load mô hình từ Prefab chứa trong resource

Để load được Prefab từ resource, ta phải đặt Prefab đó trong thư mục **Resources** của Project. Sau đó, dùng hàm ***Resources.Load()*** để load Prefab với tham số là đường dẫn của prefab tính từ thư mục Resources.

```
Obj = Resources.Load("Knights");

//... Instantiate mô hình
```

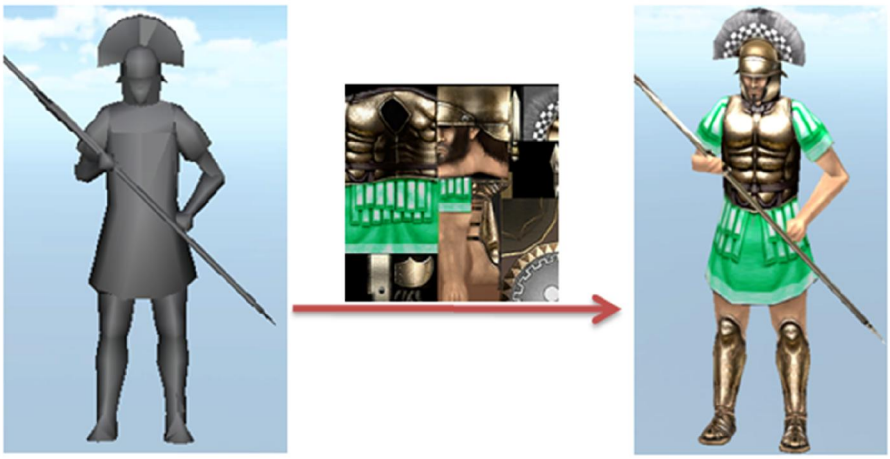


Hình 3.2 Prefab trong resources

Sau khi load được mô hình, chúng ta cần gắn texture vào mô hình để nhân vật trông bắt mắt hơn. Mô hình nhân vật không chỉ là một đối tượng game mà đôi khi còn chứa nhiều đối tượng game con bên trong, cho nên chúng ta không chỉ gán texture cho đối tượng cha mà còn gán cho tất cả đối tượng con bên trong. Chúng ta phải tìm thành phần kiểu **Renderer** (thành phần qui định cách hiển thị mô hình) trong tất cả đối tượng game để truy cập vào material của nó và gán texture vào. Đoạn code sau minh họa cách thực hiện:

Hình 3.3 Đối tượng game chứa nhiều đối tượng game con

```
Texture2D texture = (Texture2D)Resources.Load("Texture/Knights_green");

// gắn texture cho các tất cả đối tượng game
Renderer[] objRender = NewObj.GetComponentsInChildren<Renderer>();
for (int i = 0; i < objRender.Length; i++)
    objRender[i].material.mainTexture = texture;
```



Hình 3.4 Mô hình nhân vật 3D trước và sau khi gán texture

Kết luận

Rõ ràng giải pháp thứ hai cho thấy sự linh động hơn trong việc load mô hình từ Prefab, đặc biệt trong lúc runtime. Mọi thao tác biến đổi, xoay, chuyển động sau đó đều thực hiện trên đối tượng GameObject này.

3.2. Chuyển động mô hình nhân vật 3D

Vấn đề

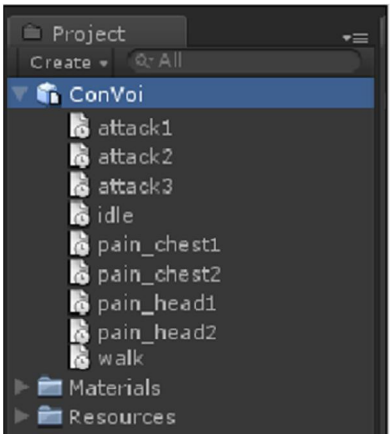
Chúng ta đã load được mô hình 3D vào trong game, vậy làm sao để mô hình 3D này có thể chuyển động trong game.

Giải pháp

Trước tiên mô hình 3D cần phải có sẵn animation bên trong. Khi import mô hình vào Unity, animation trong mô hình được tự động chuyển thành một **AnimationClip**. Điều này giúp animation này có thể dùng cho các mô hình khác trong project.

Trước hết ta phải tạo AnimationClip từ animation có sẵn của mô hình. Có 2 loại mô hình 3D có sẵn animation:

- *Loại thứ nhất:* mô hình 3D có chứa nhiều animation bên trong.



Hình 3.5 Mô hình 3D bên trong chứa nhiều animation

Mô hình 3D trên sau khi import vào project game, bên trong đã có sẵn 9 animation, mỗi animation sẽ tự động được tạo thành một AnimationClip bên trong đối tượng game.

- *Loại thứ hai:* mô hình 3D chỉ chứa một animation.

Hình 3.6 Mô hình 3D chứa một animation

Trong 3 mô hình trên, mỗi mô hình sau khi import có một AnimationClip duy nhất có tên "Default Take". Về bản chất hình dạng mô hình là như nhau, chỉ khác nhau animation (Idle, Run, Walk). Vậy làm sao chúng ta kết hợp các AnimationClip này vào một đối tượng game duy nhất. Unity quy định như sau:

- Lấy một mô hình làm mô hình chính, có thể không cần animation kèm theo cũng được.
- Các mô hình còn lại, tên phải có 2 phần cách nhau bởi '@', phần đầu phải trùng tên với mô hình chính đã chọn, phần thứ 2 sẽ là tên của animation.

Với cách đặt tên như vậy, khi đưa các mô hình này vào project để sử dụng thì Unity sẽ tự động đổi tên animation mặc định trong mô hình thành tên trùng với phần tên

mô hình nằm sau chữ '@'. Lưu ý là phải đổi tên cho mô hình từ bên ngoài project tức trên Windows vì nếu như chúng ta đổi tên trực tiếp trong project thì tên của animation của mô hình đó sẽ không bị thay đổi theo phần tên sau dấu '@'.

Hình 3.7 Hình minh họa sau khi đổi tên và import vào project

Như hình trên, mô hình sẽ có 3 AnimationClip bên trong (Default Take, idle, run).

Tiếp theo, để gọi thực hiện một AnimationClip trong một đối tượng ta dùng hàm ***CrossFade()*** của thuộc tính **animation** trong GameObject. Đoạn code sau đây đang được gắn vào đối tượng game cần chạy animation.

```
animation.CrossFade("Run"); // Run là tên của AnimationClip trong mô hình
```

Đôi khi ta muốn chạy các chuyển động khác nhau của mô hình một cách tuần tự. Để làm điều này ta dùng hàm ***CrossFadeQueued()*** của thuộc tính **animation** trong GameObject.

```
// AnimationClip Idle sẽ chạy sau khi AnimationClip Run chạy xong
animation.CrossFade("Run");
animation.CrossFadeQueued("Idle");
```

Để điều khiển cách chạy của AnimationClip, ta dùng thuộc tính **wrapMode**

```
animation.wrapMode = WrapMode.Loop; // chuyển động lặp lại liên tục
```

Kết luận

Việc gọi thực hiện các animation của đối tượng là khá đơn giản. Tuy nhiên phải quyết định chọn mô hình loại nào để có thể thêm hoặc bớt animation cho mô hình dễ dàng. Nếu chọn mô hình loại 1 thì chúng ta phải import vào các chương trình hỗ trợ làm animation cho mô hình để chỉnh sửa thêm xóa animation rồi import vào Unity lại, còn chọn mô hình loại 2 thì chúng ta chỉ cần xóa hay thêm file mô hình là

xong, rất linh hoạt và nhanh chóng. Vì vậy chúng em chọn các mô hình loại 2 để làm ứng dụng game.

3.3. Thêm sự kiện vào chuyển động của nhân vật 3D

Vấn đề

Trong lúc lập trình kịch bản game, chúng ta muốn biết khi nào một nhân vật chuyển động xong để có bước xử lý tiếp theo. Ví dụ như sau khi nhân vật thực thi chuyển động chết thì chúng ta phải hủy đối tượng đó khỏi bộ nhớ. Trong Unity, khi một chuyển động chạy xong không tự phát ra sự kiện.

Giải pháp

Unity hỗ trợ lớp **AnimationEvent** giúp thêm sự kiện vảo frame bất kỳ trong một AnimationClip. Trước hết ta phải có một file script chứa hàm sẽ thực thi sau khi sự kiện xảy ra. Chẳng hạn, file script có nội dung như sau (chứa hàm hủy đối tượng):

```
public class UnitDieCallback : MonoBehaviour
{
    void Die()
    {
        // hủy đối tượng được attach file script này
        Destroy(gameObject);
    }
}
```

Sau đó chúng ta tạo file script khác dùng để thêm event và gắn file script này vào đối tượng bất kỳ. Đoạn script mẫu bên dưới đang được gắn vào một mô hình 3D .

```
public class PlayAnimation : MonoBehaviour
{
    void Start()
    {
        AnimationEvent animEvent = new AnimationEvent();
        animEvent.functionName = "Die";
        animEvent.time = obj.animation["die"].clip.length;

        gameObject.animation["die"].clip.AddEvent(animEvent);
    }
}
```

Với đoạn code trên, chúng ta đã thêm một sự kiện vào frame cuối cùng của AnimationClip “die”. Hàm được thực thi khi sự kiện xảy ra là hàm **Die()**.

Nếu chạy đoạn code trên sẽ xảy ra lỗi thực thi vì không tìm thấy hàm Die(). Chúng ta phải thêm file script chứa hàm Die() vào đối tượng với đoạn code sau:

```
gameObject.AddComponent<UnitDieCallback>();
```

Khi chạy game, lúc đối tượng chạy animation "die" xong thì sự kiện sẽ được xảy ra. Nếu chạy animation khác thì sự kiện không xảy ra.

Kết luận

Việc thêm sự kiện vào mô hình giúp ta kiểm soát chuyển động dễ dàng hơn và đưa ra các xử lý thích hợp ở thời điểm nhất định.

3.4. Tạo địa hình trong game

Vấn đề

Trong game 3D, địa hình là thành phần không thể thiếu. Từ địa hình chúng ta có thể đặt các vật thể lên bề mặt tạo thành cảnh vật trong game. Thực chất, địa hình khi xây dựng xong là một mô hình 3D tĩnh có số lượng đa giác rất lớn cho nên người ta thường chỉ load một phần của địa hình tùy theo góc nhìn của camera.

Giải pháp

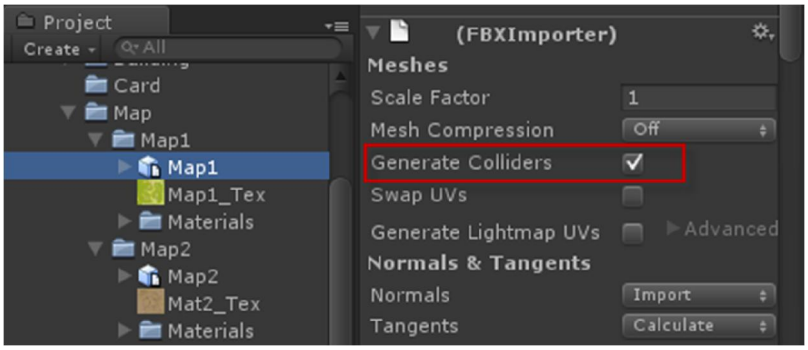
Trong Unity, có hỗ trợ sẵn công cụ để thiết kế địa hình rất mạnh và Unity API có hỗ trợ đối tượng Terrain để tạo nên địa hình trong lúc runtime. Tuy nhiên, ở thời điểm hiện tại Unity không hỗ trợ tạo địa hình trên nền Android. Cho nên, chúng em phải tạo mô hình địa hình bên ngoài trước rồi mới đưa vào engine để sử dụng. Cách thức nạp địa hình này cũng giống như chúng ta nạp mô hình 3D vậy.

```
public class LoadTerrain : MonoBehaviour
{
   public GameObject objTerrain;

   void Awake()
   {
      objTerrain = Resources.Load("Terrain");
      Instantiate(objTerrain, new Vector3(0, 0, 0), Quaternion.identity);
   }
}
```

Vì chạy trên điện thoại nên số lượng đa giác của mô hình địa hình này nên nhỏ hơn 5000 đa giác để game chạy mượt hơn.

Có rất nhiều cách để tạo mô hình địa hình, do trong Unity có sẵn công cụ tạo địa hình nên chúng em đã tận dụng công cụ này sau đó dùng một đoạn script nhỏ xuất địa hình này ra file OBJ, rồi tiếp tục dùng phần mềm thiết kế Cinema4D để tô BodyPaint texture trên mô hình địa hình vừa xuất. Cuối cùng đưa mô hình và texture trở vào engine Unity để sử dụng.

Lưu ý sau khi import mô hình địa hình vào project, trong cửa sổ Inspector của mô hình địa hình chúng ta cần check vào thuộc tính **Generate Colliders** để Unity tự động phát sinh collider (dùng để phát hiện va chạm với vật thể khác) cho mô hình khi mô hình được sử dụng trong scene. Điều này giúp chúng ta có thể đặt các mô hình nhân vật chính xác lên bề mặt địa hình này.



Hình 3.8 Check vào thuộc tính Generate Colliders

Kết luận

Do hiện tại Unity không hỗ trợ tạo địa hình lúc runtime trên nền Android nên ta phải thiết kế sẵn địa hình bên ngoài rồi đưa vào engine sử dụng.

3.5. Chiếu sáng cảnh vật

Vấn đề

Địa hình và cảnh vật trong game cũng như ngoài không gian thực cần phải có ánh sáng để mọi vật trong game rõ ràng và sáng sủa hơn.

Giải pháp

Để tạo ánh sáng mặt trời chiếu sáng toàn bộ cảnh vật ta sử dụng thành phần Light của Unity. Trước hết ta tạo một đối tượng game rồi thêm thành phần **Light** vào.

```
GameObject obj = new GameObject("Light");
Light light = obj.AddComponent<Light>();
```

Sau đó ta thay đổi loại ánh sáng của Light là **Directional**, loại ánh sáng được chiếu từ vị trí vô cực theo một hướng nhất định ảnh hưởng lên toàn bộ cảnh vật trong game, giống như mặt trời ngoài đời thực.

```
obj.transform.rotation = Quaternion.Euler(50, 180, 180); // góc chiêu
light.type = LightType.Directional;
light.color = Color.blue; // màu sắc ánh sáng
light.intensity = 3;  // mức độ sáng
```

Hình 3.9 Cảnh vật được chiếu sáng

Kết luận

Chiếu sáng làm cho địa hình nhìn thật hơn với nhiều mảng sáng tối khác nhau, mô hình nhân vật trông nổi và sâu hơn.

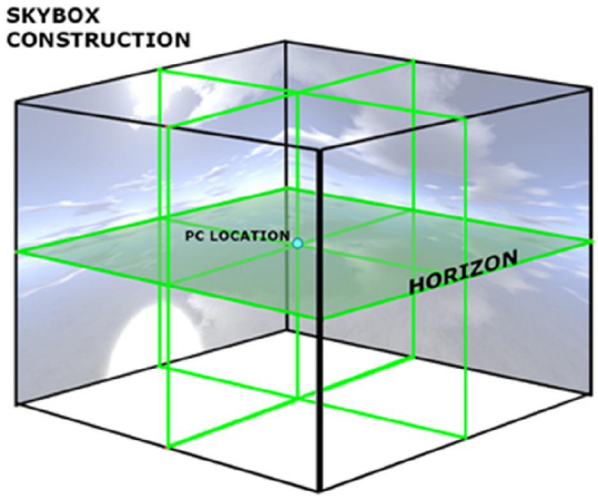
3.6. Tạo bầu trời mây

Vấn đề

Để khung cảnh xung quanh địa hình trông đẹp và thực hơn, bầu trời mây trong địa hình là cần thiết.

Giải pháp

Có nhiều cách để thực hiện bầu trời, có thể dùng kỹ thuật Skybox hoặc Skydome. Kỹ thuật Skydome có thể tạo ra bầu trời mây chuyển động trông rất sinh động. Tuy nhiên, di chuyển bầu trời liên tục làm cho game chạy không mượt lắm. Cho nên, chúng em đã dùng kỹ thuật Skybox. Skybox là kỹ thuật đặt camera ở giữa một khối lập phương mà mỗi mặt của khối lập phương này được lợp texture bên trong liên tục với nhau. Với góc nhìn phối cảnh của camera bên trong Skybox sẽ cho ta cảm giác bầu trời xa thẳm gần như thật.



Hình 3.10 Mô hình Skybox

(Nguồn: http://update.multiverse.net/wiki/index.php/Creating_Skyboxes_with_Terragen)

Skybox trong Unity là một loại chất liệu sử dụng shader "RenderFX/Skybox", chất liệu này cần 6 ảnh texture tương ứng với mỗi mặt của khối lập phương để render. Sau khi tạo chất liệu này ta dùng đoạn code sau để áp dụng hiệu ứng Skybox lên camera chính.

```
RenderSettings.skybox = (Material)Resources.Load("SunnySky");
```

Kết luận

Hiệu ứng Skybox được phối từ 6 ảnh khác nhau ghép lại sao cho hài hòa. Để tạo được 6 ảnh này ta có thể dùng công cụ Terragen của hãng Planetside software (trang chủ: http://www.planetside.co.uk/)

3.7. Tạo hiệu ứng mặt nước

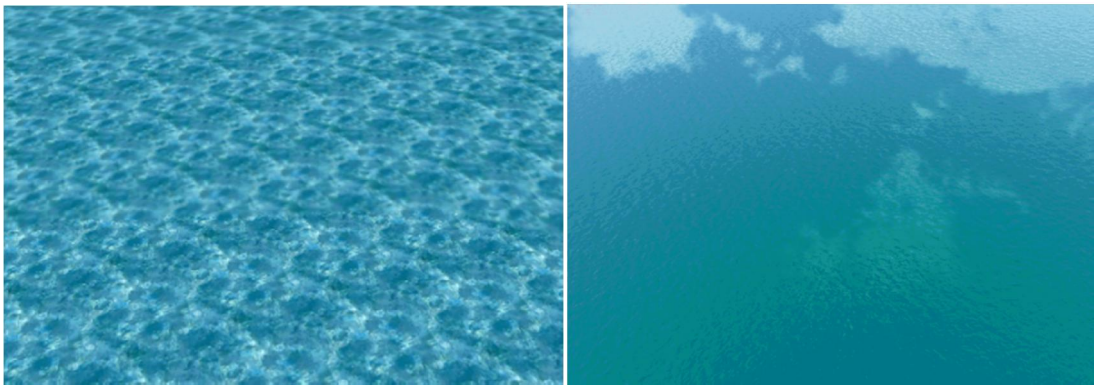
Vấn đề

Trong cảnh vật chúng ta đã có được địa hình, bầu trời. Để cảnh vật sinh động hơn nữa, chúng ta cần thêm hiệu ứng mặt nước cho cảnh vật.

Giải pháp

Trong Unity hỗ trợ Prefab mặt nước nằm trong gói "Water" đi kèm. Mặt nước này là một plane phẳng hình tròn được gắn texture mặt nước bên trên và texture này di chuyển liên tục để tạo hiệu ứng mặt nước chuyển động. Cao cấp hơn, mặt nước này còn hỗ trợ phản chiếu các cảnh vật xung quanh. Tuy nhiên, chi phí tính toán phản chiếu là khá lớn gây ra cho game hiện tượng giựt hình trên điện thoại. Khi tắt phản chiếu, nếu không quá khắt khe thì hiệu ứng mặt nước hiển thị có thể chấp nhận được. Cách load mặt nước cũng giống như cách load mô hình 3D.

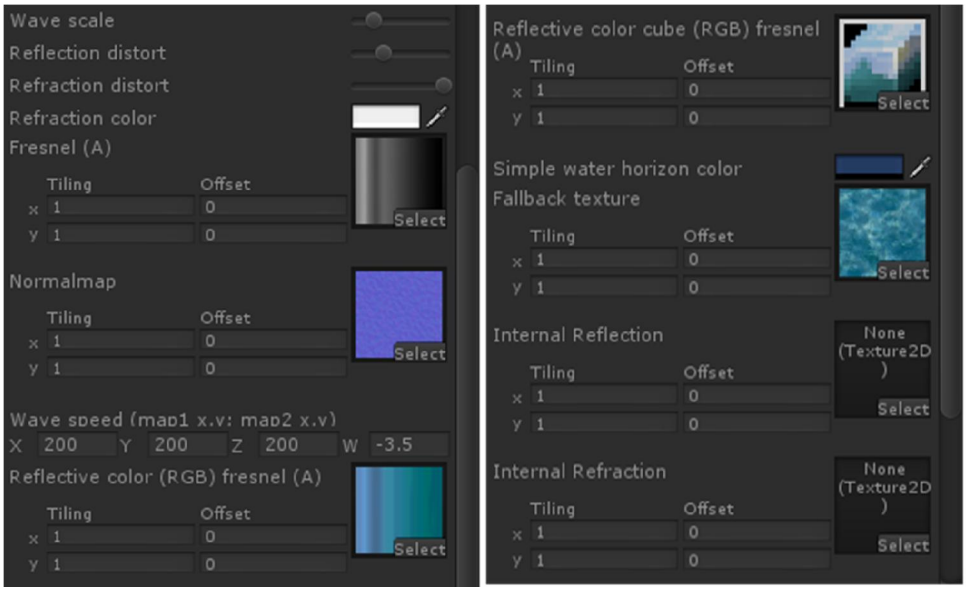
```
public class LoadWater : MonoBehaviour
{
   public GameObject objWater;

   void Awake()
   {
       objWater = Resources.Load("Daylight water");
       Instantiate(objWater, new Vector3(0, 0, 0), Quaternion.identity);
   }
}
```



Hình 3.11 Mặt nước không có phản chiếu (hình trái) và có phản chiếu (hình phải)

Chúng ta có thể thay đổi các thuộc tính của shader mặt nước như:

- **Wave speed**: tốc độ của sóng
- **Wave scale**: độ lớn của sóng
- **Fallback texture**: vân bề mặt sóng



Hình 3.12 Các thuộc tính của Shader tạo mặt nước

Kết luận

Bằng cách sử dụng mặt nước có sẵn của Unity kết hợp tùy chỉnh các thông số chúng ta sẽ có được mặt nước sống động hơn.

3.8. Đặt mô hình 3D lên địa hình

Vấn đề

Sau khi đã có địa hình đẹp mắt, ta cần đặt nhân vật lên trên địa hình không bằng phẳng. Việc đặt một mô hình vào game thì đơn giản nhưng để đặt mô hình đúng trên bề mặt địa hình cũng là một vấn đề cần quan tâm.

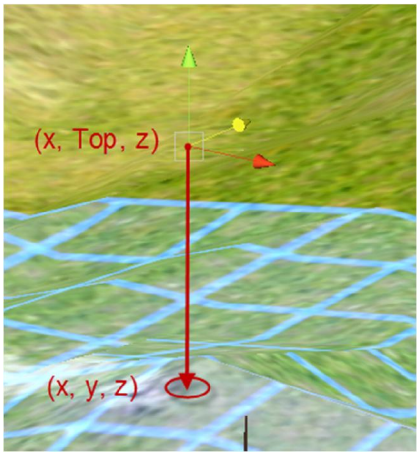
Giải pháp

Muốn đặt nhân vật lên địa hình tại một vị trí (x, y, z), ta phải tính được vị trí y của mô hình muốn đặt sao cho theo độ cao của địa hình tại vị trí đó.

Để làm điều này, trước hết ta phải biết vị trí y cao nhất có thể của địa hình, ta gọi đó là Top (giả sử địa hình đặt ở vị trí y=0).

Sau đó sử dụng phương thức ***Physics.Raycast()*** để chiếu một tia từ vị trí (x, Top, z) xuống bên dưới địa hình, kết quả ta sẽ nhận được điểm mà tia chạm với địa hình. Điểm chạm đó cũng chính là nơi ta đặt nhân vật. Đoạn code sau minh họa điều này (đoạn code này đang được gắn vào mô hình nhân vật):

```
Vector3 AbovePoint = new Vector3();  // điểm bên trên điểm cần đặt
AbovePoint.x = x;
AbovePoint.y = Top;
AbovePoint.z = z;

// chiếu 1 tia thẳng xuống địa hình
RaycastHit hit;
if (Physics.Raycast(AbovePoint, Vector3.down, out hit, Top))
   transform.position = hit.point; // đặt mô hình tại điểm chạm
```



Hình 3.13 Chiếu Raycast xuống địa hình để tìm điểm chạm trên bề mặt

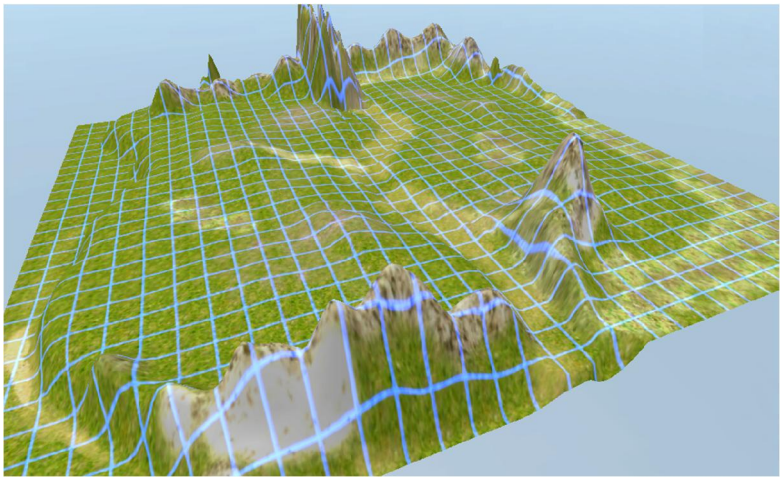
Kết luận

Bằng cách dùng Physics.Raycast(), ta đã đặt được nhân vật ở vị trí mong muốn. Ta cũng có thể dùng cách này để đặt cây cối lên địa hình làm cho địa hình thật hơn nữa.

3.9. Vẽ lưới trên địa hình không bằng phẳng

Vấn đề

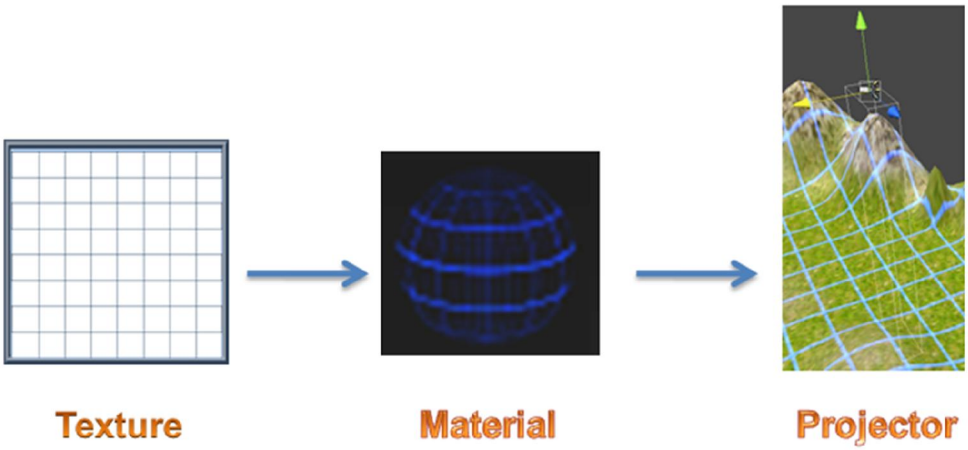
Trong thể loại game chơi theo lượt như game Heroes of Might and Magic, mỗi nhân vật phải di chuyển trong một lưới được định sẵn trên địa hình. Chúng ta cần vẽ lưới này trên địa hình để giúp người chơi dễ dàng xác định vị trí cần di chuyển đến. Trong game 2D, lưới này rất dễ hiển thị trên địa hình do địa hình 2D là địa hình phẳng. Tuy nhiên, trong game 3D, địa hình không phải lúc nào cũng bằng phẳng, vấn đề là làm sao chúng ta có thể vẽ những đường thẳng được bám theo độ cao của địa hình.



Hình 3.14 Lưới vẽ bám theo độ cao của địa hình

Giải pháp

Để giải quyết vấn đề này, chúng em sử dụng **Projector**. Tư tưởng ở đây là chúng ta sẽ vẽ một hình lưới, sau đó tạo chất liệu từ hình lưới này và shader thích hợp. Tiếp theo ta sẽ dùng Projector chiếu song song chất liệu này xuống bề mặt địa hình.



Hình 3.15 Qui trình vẽ lưới trên địa hình

Ta sử dụng lớp **Texture2D** để tạo texture. Dùng hàm ***SetPixel()*** để vẽ từng điểm trên texture tạo thành các đường thẳng rồi gọi ***Apply()*** để áp dụng thay đổi vừa vẽ lên texture.

```
Texture2D texture = new Texture2D(TextureWidth, TextureHeight);

// vẽ 1 đường thẳng
```

```
for (int i = 0; i < TextureWidth; i++)
   texture.SetPixel(i, 0, Color.white);

// vẽ tiếp các đường thẳng tiếp theo ...

// áp dụng tác dụng của SetPixel lên texture
texture.Apply();
```

Tiếp theo ta dùng shader "**Mobile/Particles/Additive Culled**" trong gói "Standard Assets (Mobile)" đi kèm theo Unity để tạo chất liệu cho Projector từ Texture vừa vẽ. Shader là một đoạn script nhỏ thiết lập cách thức render của chất liệu trên bề mặt. Đoạn code sau dùng để tạo material từ shader và gán texture cho chất liệu:

```
Shader shader = Shader.Find("Mobile/Particles/Additive Culled");
Material material = new Material(shader);


material.mainTexture = texture;
```

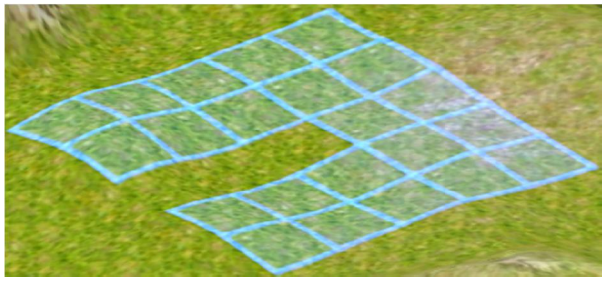
Sau khi có chất liệu, ta tạo Projector và áp chất liệu này vào.

```
GameObject obj = new GameObject("Projector");


Projector projector = obj.AddComponent<Projector>();
projector.material = material;
```

Cuối cùng, ta di chuyển Projector đến vị trí của địa hình và chiếu song song xuống địa hình.

```
// di chuyển
projector.transform.position = new Vector3(20, 20, 20);

// xoay
projector.transform.rotation.eulerAngles  = new Vector3(-90, 0, 0);

// chiếu song song
projector.isOrthoGraphic = true;
```

Do texture được vẽ lúc runtime nên chúng ta có thể mở rộng chức năng chỉ vẽ một số ô nào đó trên địa hình.



Hình 3.16 Vẽ lưới trên một phần của địa hình

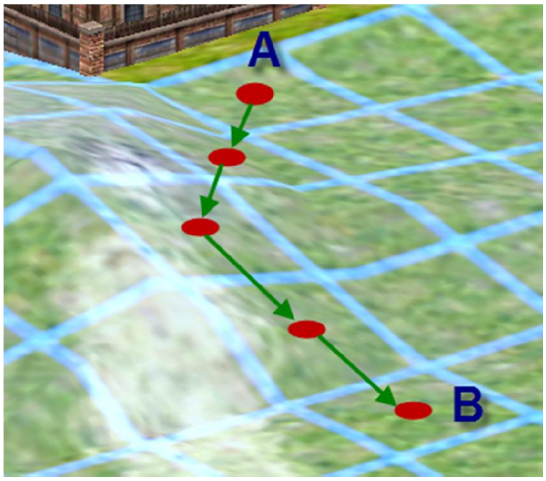
Kết luận

Trong quá trình tìm hiểu để giải quyết vấn đề này, chúng em chưa tìm thấy hướng dẫn cụ thể nào. Trên đây là một cách của chúng em đưa ra để giải quyết vấn đề đã nêu.

3.10. Xử lý di chuyển trong bản đồ

Vấn đề

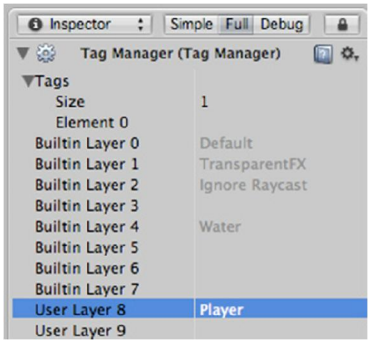
Trong thể loại game chơi theo lượt, quân lính phải có khả năng di chuyển từ vị trí A đến vị trí B bằng con đường ngắn nhất. Đồng thời lúc mô hình di chuyển từ A sang B trên địa hình 3D phải đi theo độ cao của địa hình chứ không được đi thẳng.



Hình 3.17 Đường đi từ ô A sang ô B trên địa hình lưới

Giải pháp

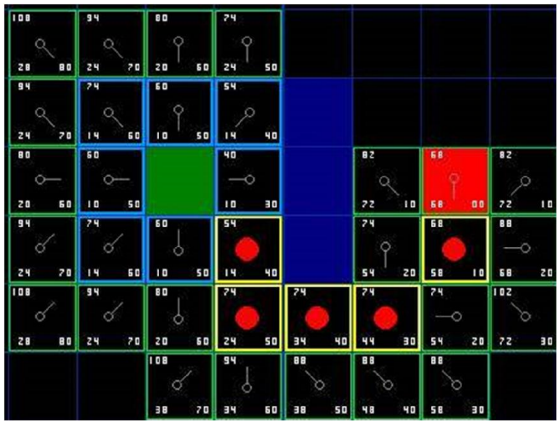
Trước khi vào vấn đề, chúng ta cần hiểu sơ về **Layer** trong Unity. Có thể hiểu layer tương tự như khái niệm nhóm. Mỗi đối tượng game trong Unity đều thuộc về một layer nào đó, giúp ta có thể xử lý chung cho các đối tượng thuộc cùng layer mà không cần xử lý riêng lẻ. Chúng ta có thể thêm layer mới bằng cách vào menu **Edit → Project Settings → Tags**. Các layer được đánh số từ 0 kèm tên layer tương ứng.



Hình 3.18 Màn hình quản lý Layer

Trong hàm ***Physics.RayCast()***, layer được biểu diễn dưới dạng bitmask, vị trí bit X có giá trị bằng 1 có nghĩa layer X đang bật, bằng 0 là không được bật. Ví dụ để trỏ đến layer 8 (layer Player) ở hình 3.17, ta dùng 1<<8.

Để di chuyển từ ô A đến ô B trong bản đồ lưới, chúng em dùng thuật toán A* để tìm đường đi ngắn nhất giữa 2 ô, có kiểm tra các chướng ngại vật.



Hình 3.19 Thuật toán A* tìm đường đi ngắn nhất giữa 2 ô

(Nguồn: http://www.policyalmanac.org/games/aStarTutorial.htmb)

Sau khi có được con đường ngắn nhất là vị trí các ô cần đi, chúng ta chuyển các ô đó sang vị trí thực trên địa hình. Lưu ý vị trí y của các điểm phải theo độ cao của địa hình tại đó, có thể dùng Raycast để làm điều này (tương tự như phần 3.8).

Để di chuyển mô hình từ vị trí X đến vị trí Y mượt mà và không có sự gãy khúc, ta sử dụng thư viện **iTween** – thư viện giúp tạo chuyển động cho các đối tượng game trong Unity. Thư viện iTween rất mạnh trong việc xử lý di chuyển, xoay, phóng to, thu nhỏ, … Hơn thế nữa, iTween hỗ trợ khá nhiều các hiệu ứng chuyển động rất mượt và đẹp mắt. Để di chuyển từ vị trí hiện tại đến vị trí Y với iTween, ta dùng đoạn code sau (đoạn code này đang được gắn vào mô hình cần di chuyển):

```
//di chuyển gameObject từ vị trí hiện tại đến vị trí Y trong thời gian 1s
iTween.MoveTo(gameObject, Y, 1F);
```

Để di chuyển qua toàn bộ vị trí, ta dùng đoạn code sau:

```
// di chuyển gameObject đến tất cả vị trí trong thời gian 20s
Hashtable args = new Hashtable();
args.Add("path", path); // path là mảng Vector3 chứa các điểm cần đi qua
args.Add("speed", 20F); // tốc độ di chuyển
iTween.MoveTo(gameObject, args);
```

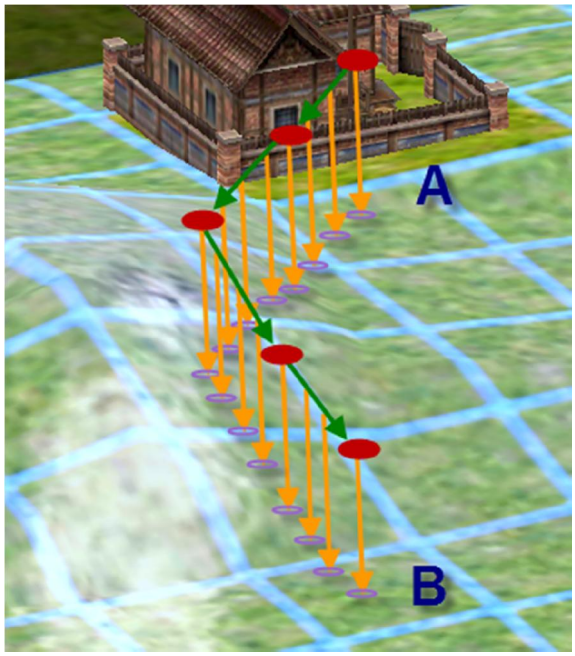
Trong iTween mỗi hàm chuyển động khi chuyển động xong đều phát ra sự kiện báo hiệu đã hoàn thành. Lợi dụng điều này, trước lúc di chuyển ta có thể cho mô hình nhân vật chạy animation đi liên tục và khi đi đến đích thì nhân vật animation đứng im. Ta sẽ thêm tham số **onComplete** vào mảng tham số với giá trị là hàm cần chạy khi hoàn thành.

```
args.Add("onComplete", "PlayIdle");
args.Add("onCompleteTarget", gameObject);
```

Trong đoạn code trên có một tham số quan trọng là **onCompleteTarget**, tham số này trỏ đến đối tượng đang được gắn file script mà chứa hàm ***PlayIdle()*** bên trong.

Đến đây nhân vật đã di chuyển rất mượt qua các vị trí trên địa hình.

Tuy nhiên, địa hình giữa 2 vị trí không phải lúc nào cũng bằng phẳng, ta phải làm sao để lúc di chuyển mô hình phải thay đổi vị trí y theo độ cao địa hình. Để giải quyết vấn đề này, ta tạo một đối tượng game ẩn bên trên mô hình quân lính và thay vì cho mô hình di chuyển thì ta cho đối tượng game này di chuyển qua toàn bộ vị trí. Đồng thời, trong lúc di chuyển, đối tượng game này sẽ dùng Raycast chiếu thẳng xuống địa hình và đặt quân lính tại vị trí Raycast chạm với địa hình.



Hình 3.20 Vừa di chuyển vừa chiếu Raycast xuống địa hình

Đoạn code sau sẽ thực hiện điều này:

```
using UnityEngine;

public class MoveObject : MonoBehaviour
{
   GameObject Followter;

   void Start(){
      // tim đường đi ngắn nhất giữa 2 ô -> path ....

      // tạo GameObject Followter bên trên GameObject hiện tại
      Vector3 FollowterPosition = gameObject.transform.position;
      FollowterPosition.y += 30;

      Followter = new GameObject("Followter");
      Followter.transform.position = FollowterPosition;
```

```
      // animation đi ...

      // di chuyển Followter qua các vị trí trong thời gian 20s
      Hashtable args = new Hashtable();
      args.Add("path", path);
      args.Add("speed", 20F);
      args.Add("onComplete", "PlayIdle");
      args.Add("onCompleteTarget", gameObject);

      iTween.MoveTo(Followter, args);
   }

   void Update (){
      RaycastHit hit;
      if (Physics.Raycast(Followter.transform.position, Vector3.down, out
hit, 30, 1 << 8)) // layer 8 là layer của địa hình
         gameObject.transform.position = hit.point;
   }

   void PlayIdle (){
      // animation đứng im ...
   }
}
```

Kết luận

Kết hợp thư viện chuyển động iTween với Raycast xuống địa hình giúp mô hình di chuyển trông rất thật trên bề mặt địa hình.

3.11. Tạo hiệu ứng particle

Vấn đề

Các hiệu ứng thường gặp trong game như mưa, tuyết rơi, khói, lửa, hiệu ứng phép,… sẽ làm cho game sinh động và ấn tượng hơn, nhất là với game 3D thì các hiệu ứng này càng cần thiết hơn. Các hiệu ứng này được gọi chung là hiệu ứng particle.

Giải pháp

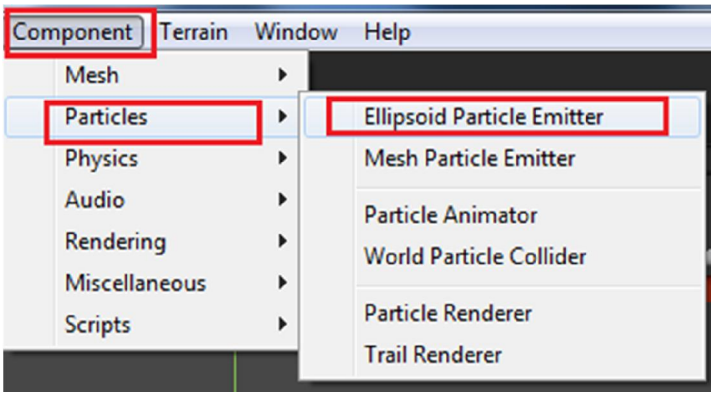
Để làm được điều này, Unity hỗ trợ người dùng Particle Systems để tạo ra bất kỳ hiệu ứng particle nào mà người dùng mong muốn. Particle muốn hiển thị được phải có 3 thành phần chính quan trọng sau:

- **Particle Emitter**: để sinh ra các hạt.
- **Particle Animator**: để làm di chuyển các hạt theo thời gian.
- **Particle Renderer**: để vẽ các hạt.

Để tạo một particle, chúng ta thực hiện như sau:

► *Bước 1:*

Tạo thành phần quan trọng nhất để sinh ra các hạt - thành phần **Particle Emitter**. Thành phần này không thể tạo trực tiếp từ code mà chỉ có thể thêm từ giao diện Editor của Unity bằng cách chọn menu **Component → Particles → Ellipsoid Particle Emiter**.



Hình 3.21 Thêm thành phần Ellipsoid Particle Emitter

► *Bước 2:*

Tạo 2 thành phần còn lại là **Particle Animator** và **Particle Renderer**. Bước này chúng ta có thể thực hiện bằng code hoặc trên giao diện.

```
public class Particle : MonoBehaviour
{
  void Start ()
  {
    //tạo thành phần ParticleRenderer để vẽ các hạt
    ParticleRenderer pRen = gameObject.AddComponent<ParticleRenderer>();
    //tạo thành phần ParticleAnimator để chạy các hạt
```

```
    ParticleAnimator pAmin = gameObject.AddComponent<ParticleAnimator>();
  }
}
```

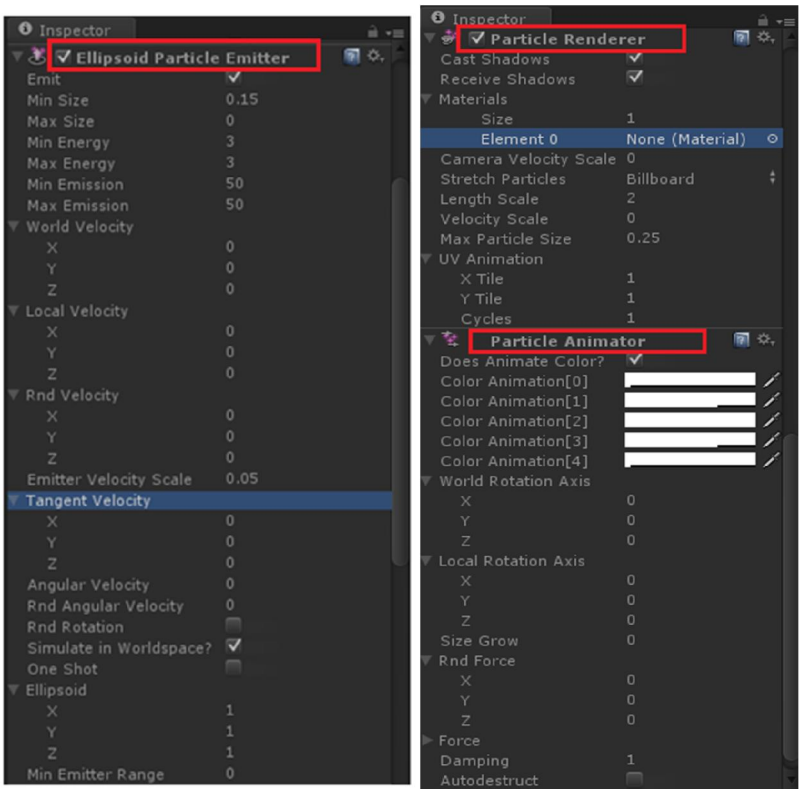
Phương thức ***AddComponent<ParticleRenderer>()*** để thêm một thành phần **Particle Renderer** vào gameObject. Tương tự như vậy với thành phần **Particle Animator**. Cách lấy và gán thuộc tính thông qua biến trả ra của hàm AddComponent().

Để gọi và thay đổi các thuộc tính của thành phần **Particle Emiter** chúng ta sử dụng thuộc tính **particleEmiter** của GameObject:

```
particleEmitter.maxSize = 1F;
particleEmitter.minSize = 0.15F;
```

- *Bước 3:*

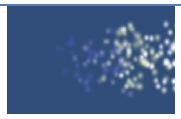
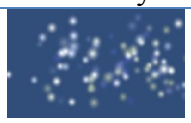
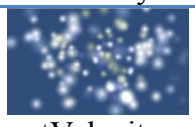
Tùy chỉnh các thuộc tính riêng của từng thành phần để có được một hiệu ứng như mong muốn. Xem qua các thuộc tính của 3 thành phần trên giao diện editor để thấy rõ hơn:



Hình 3.22 Các thuộc tính của Particle System

✓ Các thuộc tính của thành phần tạo hạt, **Elipsoid Particle Emiter** được trình bày trong bảng 3.1

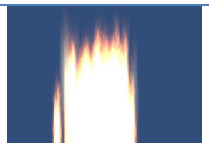
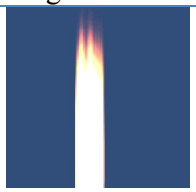
Bảng 3.1 Các thuộc tính của Elipsoid Particle Emiter

Thuộc tính	Ý nghĩa	Hình minh họa
Emit	Nếu enable hiệu ứng sẽ phát ra.	 Mặc định
Min Size / Max Size	Kích thước nhỏ nhất/lớn nhất có thể của mỗi hạt tại thời điểm sinh ra.	 Max Size = 0.7
Min Energy / Max Energy	Thời gian sống nhỏ nhất/lớn nhất của hạt, tính bằng giây.	 Max Energy = 7
Min Emisson / Max Emisson	Số lượng tối thiểu/tối đa của hạt được phát ra, tính bằng giây.	 Max Emission = 1
World Velocity	Tốc độ bắt đầu của hạt theo các chiều x, y, z trong không gian.	 WorldVelocity.x = 1
Local Velocity	Tốc độ bắt đầu của các hạt trong cùng một vùng x, y, z	 LocalVelocity.x = 1
Rnd Velocity	Vận tốc ngẫu nhiên của các hạt cùng chiều x, y, z.	 RndVelocity.x=3
Tangent Velocity	Vận tốc khởi đầu cho các hạt cùng chiều trên bề mặt của Emitter.	 TangentVelocity.x= 0.5
OneShot	Nếu enable thì hiệu ứng hiện một lần rồi tắt, sau đó hiện lên lại. Nếu tắt thì hiệu ứng hiện liên tục.	Hiện và ẩn cả khối hiệu ứng

EllipsoidScale	Phép tỷ lệ của khối hiệu ứng.	 Tỷ lệ (2,1,1)

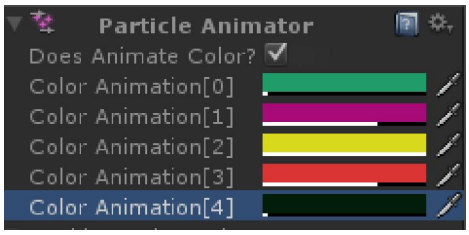
✓ Các thuộc tính của thành phần vẽ, **ParticleRenderer** được trình bày trong bảng 3.2

Bảng 3.2 Các thuộc tính của ParticleRenderer

Thuộc tính	Ý Nghĩa	Hình minh họa
Materials	Chất liệu của particle, có thuộc tính size và texture, texture để chứa các texture bên ngoài đưa vào giúp hiệu ứng đẹp hơn.	Texture:  
Stretch Particles	Độ căng giản của các hạt.	 StretchParticles= HorizontalBillboard
Length Scale	Tỷ lệ độ dài của các hạt	 StretchParticles= Stretched, LengthScale=10
Velocity Scale	Vận tốc của các hạt nếu dựa vào thuộc tính "*Stretch Particles*".	 Velocity Scale =15

✓ Các thuộc tính của thành phần làm chuyển động, **ParticleAnimator** được trình bày trong bảng 3.3

Bảng 3.3 Các thuộc tính của Particle Animator

Thuộc tính	Ý Nghĩa	Hình minh họa
Color Animation	Màu sắc của hạt, chu kỳ màu của hạt sẽ mất nhanh nếu hạt nào có tốc độ nhanh hơn. 	 DoesAnimateColor= true
Does Animate Color	Nếu không bật thì các chu kỳ màu loang sẽ mất, nếu bật thì các chu kỳ màu loang sẽ có tác dụng.	 DoesAnimateColor= false
World Rotation Axis/ Local Rotation Axis	Phép quay quanh các trục x, y, z.	 WorldRotationAxis.x =5
Force	Tưởng tượng như có một sức gió đang thổi vào các hạt theo các chiều x, y, z.	 Force.x =-1
Damping	Sự tắt dần chuyển động của từng hạt.	 Damping.x=2.4

Kết luận

Nắm rõ các thuộc tính của 3 thành phần chính của Particle System trong Unity và biết cách áp dụng một cách linh hoạt, chúng ta hoàn toàn có thể tạo ra các hiệu hạt khác nhau và đẹp mắt để đưa vào game của mình.

3.12. Xây dựng giao diện game

Vấn đề

Giao diện đồ họa người dùng là một phần quan trọng không thể thiếu trong khi xây dựng một ứng dụng game hay bất cứ một ứng dụng nào để vẽ các đối tượng đồ họa như Button, Label, Checkbox, Slider, … lên màn hình.

Giải pháp

Để làm được điều này chúng ta dùng lớp GUI, GUI là chữ viết tắt của "Graphical User Interface" – "Giao diện đồ họa người dùng". Hệ thống GUI của Unity được gọi là GUIUnity. Để sử dụng được các phương thức trong GUI ta phải gọi thực hiện từ trong hàm **OnGUI()** giống như sự kiện Paint trong C#. Ví dụ sau đây sẽ tạo ra một button đơn giản:

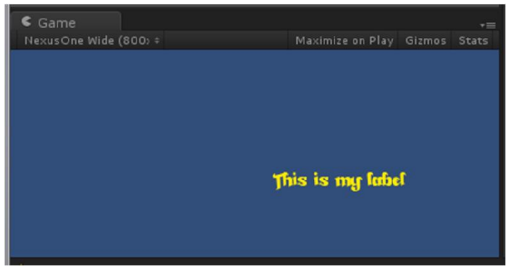
```
void OnGUI()
{
     if (GUI.Button(new Rect(10, 10, 150, 100), "I am a button"))
     {
         print("You clicked the button!");
     }
}
```

Hình 3.23 GUI Button

Để tạo một Button hay một thành phần trong GUI đẹp hơn, chúng ta sử dụng **GUIStyle** - style cài đặt cho các thành phần của GUI. GUIStyle chứa thông tin về font chữ, vị trí đặt biểu tượng, hình nền, khoảng cách, … Chỉ cần định nghĩa và áp dụng cho một bất kỳ thành phần nào như Button, Label, Checkbox,… Ta có thể xem GUIStyle như CSS khi thiết kế web site vậy.

Ví dụ sau tạo một Label theo một định nghĩa GUIStyle, với kích thước font là 14, chữ in đậm, canh lề chữ ở giữa, màu chữ đỏ, font Beckasin (được lưu trước trong thư mục Resources), vị trí vẽ ra (Screen.width/2, Screen.height/2), kích thước hình chữ nhật bao quanh 150x50:

```
void OnGUI()
{
   GUIStyle myStyle = new GUIStyle();
   myStyle.fontSize = 14;
   myStyle.fontStyle = FontStyle.Bold;
   myStyle.alignment = TextAnchor.MiddleCenter;
   myStyle.normal.textColor = Color. yellow;
   myStyle.font = (Font)Resources.Load("Fonts/Beckasin");
   GUI.Label(new Rect(Screen.width/2,Screen.height/2, 150,50),"This is my label",myStyle);
}
```

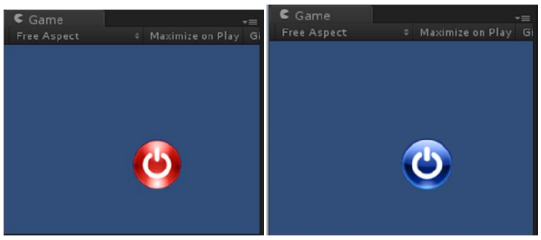


Hình 3.24 Áp dụng GUIStyle lên Label

Ví dụ sau đây sẽ vẽ ra một Button với background là hình. Vì chỉ vẽ hình nên chúng ta không cần quan tâm đến định dạng text trong GUIStyle nữa, chúng ta chỉ quan tâm đến hình nền của button với các sự kiện chuột tương tác trên button này mà thôi.

```
void OnGUI()
{
     GUIStyle myStyle = new GUIStyle();

     myStyle.normal.background = (Texture2D)Resources.Load("Button/exit");
     myStyle.hover.background = (Texture2D)Resources.Load("Button/exit_hover");

     if (GUI.Button(new Rect(Screen.width / 2, Screen.height / 2, 50, 50), "My button", myStyle)) {
```

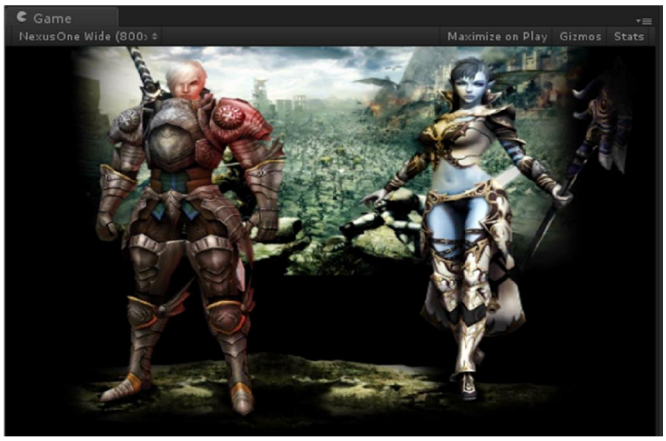
```
        //do something....
    }
}
```



Hình 3.25 Button khi rê chuột và không rê chuột

Để vẽ một Image ra màn hình, chúng ta sử dụng Texture2D, nó tương tự đối tượng Bimap hay Image bên ngôn ngữ C#. Để load hình từ Resources và vẽ ra màn hình chúng ta dùng phương thức **DrawTexture()** như sau:

```
Texture2D myImage = (Texture2D)Resources.Load("background_image");
GUI.DrawTexture(new Rect(0,0,Screen.width,Screen.height), myImage);
```



Hình 3.26 Vẽ hình ảnh trên GUI

❖ **Kết luận**

Với lớp GUI trong Unity, chúng ta hoàn toàn có xây dựng nên một giao diện tuyệt vời cho ứng dụng game. Ngoài các phương thức của lớp GUI đã nêu trên thì còn rất nhiều phương thức vẽ các đối tượng khác như Radio, Checkbox, Slider…

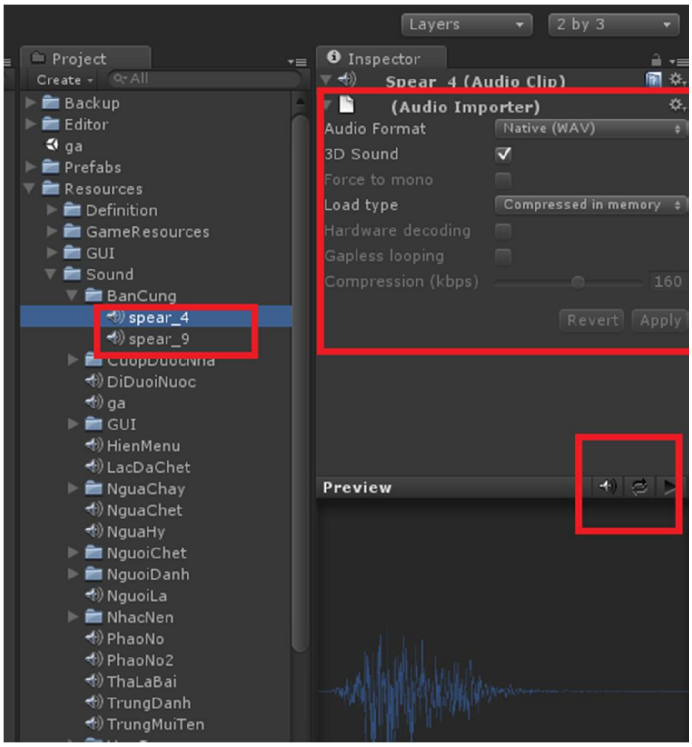
3.13. Âm thanh trong game

Vấn đề

Âm thanh là yếu tố không kém phần quan trọng trong ứng dụng game. Thật nhàm chán khi một cảnh đánh đánh nhau, bắn nhau hay các hiệu ứng đẹp mắt mà không có âm thanh. Âm thanh 3 chiều sẽ làm cho game thực hơn và sống động hơn.

Giải pháp

Để chơi được một file âm thanh trong Unity có 2 cách: bằng code hoặc trên giao diện. Dù chọn cách nào thì trước hết chúng ta phải có sẵn các file âm thanh và import vào project. Sau khi import âm thanh vào project, nếu file hợp lệ chúng ta sẽ thấy như hình sau và có thể nhấn nút play để nghe thử.

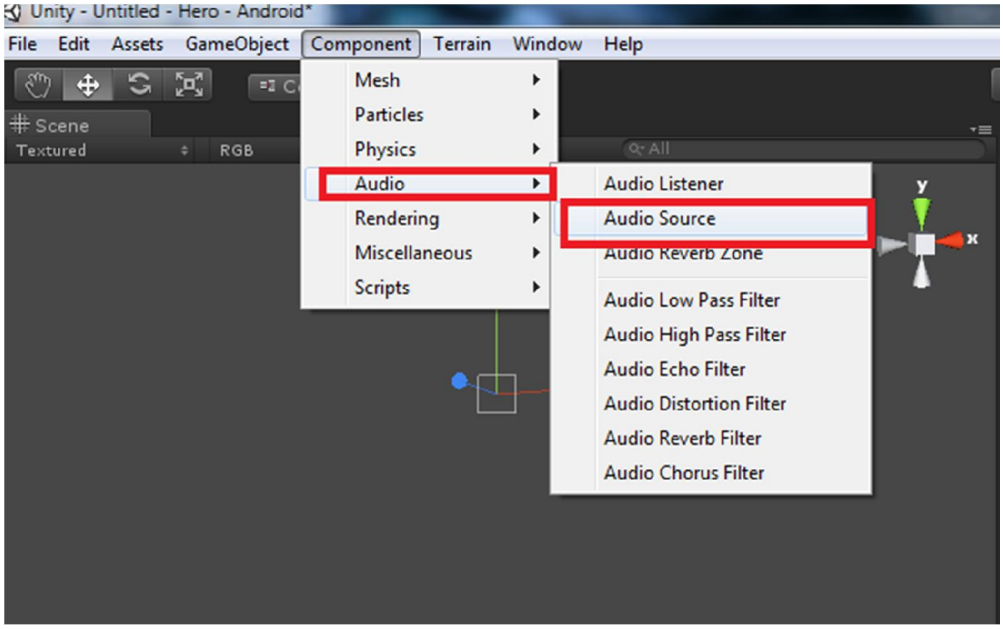


Hình 3.27 Thông tin file âm thanh

- *Cách 1:* Tạo trên giao diện

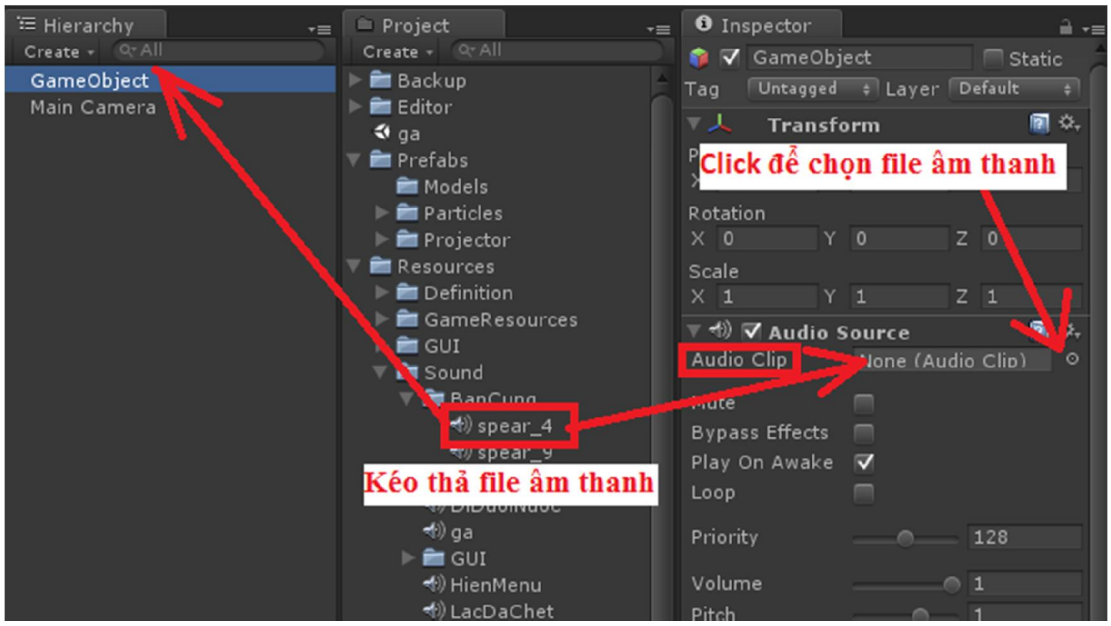
Trên menu của Unity, vào **GameObject → Create Empty**.

Chọn đối tượng vừa tạo và gắn thành phần "**AudioSource**" cho đối tượng này. AudioSource là một đối tượng âm thanh. Muốn Play hay Stop, thay đổi cách lặp, tăng giảm volume nhạc thì phải thông qua đối tượng này.



Hình 3.28 Thêm thành phần Audio Source

Sau khi gắn thành phần âm thanh cho đối tượng vừa tạo, chúng ta dễ dàng chỉnh sửa các thông số và gán file âm thanh cho thành phần AudioSource này.



Hình 3.29 Thêm file âm thanh cho thành phần AudioSource

Nếu thực hiện xong các bước trên, chúng ta có thể chạy game để nghe thử. Việc tạo đối tượng âm thanh trên giao diện khá đơn giản, nhưng để áp dụng vào cho game thì không được linh hoạt bằng cách sử dụng script.

- *Cách 2:* Cách chơi nhạc bằng code

Trước tiên chúng ta cần import file âm thanh vào project trước.

Khởi tạo đối tượng game âm thanh **AudioSource** như sau:

```
AudioSource audioSource =
(AudioSource)gameObject.AddComponent("AudioSource");
```

Gán đường dẫn file nhạc cho âm thanh:

```
audioSource.clip = (AudioClip)Resources.Load("TenFileAmThanh");
```

Sau đó chỉ cần gọi các phương thức Play() để chạy file âm thanh:

```
//play nhạc 1 lần
audioSource.PlayOneShot();
//play nhạc lặp đi lặp lại
audioSource.loop = true;
audioSource.Play();
//stop chơi nhạc
audioSource.Stop();
```

Kết luận

Để chơi được âm thanh trong Unity thì chỉ cần áp dụng các kỹ thuật nêu trên là đủ. Ngoài ra còn nhiều thành phần khác như: AudioListener, AudioSetting để tạo hiệu ứng âm thanh 3 chiều thực hơn cho game.

Chương 4

Ứng dụng game phát triển trên Unity

✍Nội dung chương này mô tả các khái niệm, quy luật chơi trong dòng game theo lượt mà chúng em xây dựng, đồng thời nêu lên kiến trúc của game.

4.1. Giới thiệu game

Đặc trưng của dòng game theo lượt chúng em xây dựng là tại một thời điểm chỉ có một người chơi có thể điều khiển quân của mình. Một màn chơi có thể bắt đầu với một số nhóm và điều kiện chiến thắng khác nhau. Các nhóm này có thể là bạn hay kẻ thù của nhau. Mỗi nhóm ban đầu sẽ có một anh hùng và một số lượng lính nhất định. Tất cả quân lính của các nhóm đều nằm trên một bản đồ chiến thuật. Người chơi sẽ điều khiển từng quân lính của mình thực hiện hành động mong muốn.

Trong một màn chơi, trên vùng chơi sẽ có nhiều công trình khác nhau. Mỗi công trình lúc đầu không thuộc về một phe nào cả. Người chơi phải chiếm đóng để tăng vàng, cứu chữa cho lính của mình và mua lính mới.

Sau khi kết thúc lượt đi của mình, người chơi sẽ chuyển lượt cho các phe khác. Sau khi các phe khác đi xong sẽ tiếp tục lượt của người chơi. Người chơi thua cuộc khi anh hùng bị chết.

Hình 4.1 Game chúng em xây dựng có lối chơi gần tương tự game Fantasy war

4.2. Các qui luật chơi chính

4.2.1. Di chuyển

Mỗi quân lính sẽ có vùng di chuyển khác nhau. Trong một lượt, mỗi quân lính chỉ được di chuyển một lần. Sau khi di chuyển, nếu có kẻ thù xung quanh phạm vi đánh của quân lính đó thì quân lính đó được phép tấn công kẻ thù.

Hình 4.2 Khu vực có thể di chuyển của quân lính

4.2.2. Tấn công

Tùy theo phạm vi đánh của quân lính mà ta chia ra 2 loại tấn công: đánh và bắn. Khi một quân lính đánh kẻ thù thì kẻ thù sẽ tự động đánh lại quân lính đó. Nếu quân lính bắn kẻ thù thì kẻ thù không thể bắn lại. Tùy theo quân lính mà ta sẽ có chiến thuật đánh khác nhau. Khi tấn công xong thì quân lính đó hết lượt. Số lượng máu đối phương giảm khi bị đánh được tính bằng công thức sau:

$$\text{Số máu B bị mất khi A đánh B} = \frac{\text{Atk (A)}}{\text{Def (B)}} \times \frac{\text{HP(A) hiện tại}}{\text{HP(A) gốc}} \times 5$$

Do đó lượng máu hiện tại của quân lính càng thấp thì càng yếu. Nhưng đối với anh hùng thì không bị ảnh hưởng bởi yếu tố máu này.

Hình 4.3 Tấn công trong game

4.2.3. Cứu chữa

Do yếu tố máu ảnh hưởng đến sức đánh của quân lính nên quân lính sẽ có khả năng được cứu chữa. Một quân lính muốn được cứu chữa thì phải đứng cạnh công trình mà người chơi đang sở hữu. Phí cứu chữa của từng loại quân lính là khác nhau.

4.2.4. Nâng cấp kỹ năng

Mỗi quân lính sẽ tăng điểm kinh nghiệm sau khi tấn công kẻ thù. Sau khi đầy điểm kinh nghiệm, quân lính đó sẽ được thêm 1 điểm kỹ năng. Điểm kỹ năng dùng cho quân lính nâng cấp kỹ năng mong muốn. Mỗi cấp độ của quân lính sẽ có những kỹ năng được nâng cấp khác nhau.

Hình 4.4 Bảng nâng cấp kỹ năng của quân lính

4.2.5. Tài nguyên

Tài nguyên sử dụng trong game là vàng. Vàng dùng để mua lính và phục hồi máu cho quân lính. Số vàng được tự động tăng qua mỗi lượt tùy theo số công trình mà người chơi đang sở hữu.

4.3. Các khái niệm trong game

4.3.1. Bản đồ chiến thuật

Người chơi sẽ chơi trên một bản đồ chiến thuật dạng lưới ô vuông. Mỗi ô sẽ có loại địa hình khác nhau và quân lính sẽ được nhận thêm bonus của địa hình khi đứng trên đó. Ví dụ quân bắn cung khi bước vào ô có loại địa hình là đồi núi thì tầm bắn sẽ tăng lên, quân kị binh đi xuống sông thì sức đỡ sẽ bị giảm, ...

Hình 4.5 Bản đồ chiến thuật

4.3.2. Người chơi

Người chơi trong game có nhiệm vụ di chuyển, đánh, mua binh lính, chiếm các công trình, … thực hiện các chiến lược để tiêu diệt kẻ thù. Có 2 loại người chơi : người chơi do con người điều khiển và người chơi do máy điều khiển. Người chơi do máy điều khiển có bổ sung trí tuệ nhân tạo để tự động thực hiện các công việc vừa nêu.

Bắt đầu màn chơi, người sẽ được cấp một anh hùng và một số lính nhất định. Người chơi điều khiển nhân vật của mình đi chiếm các công trình để có vàng và mua binh lính chiến đấu để có thể thắng các đối thủ trong màn chơi đó.

4.3.3. Công trình

Công trình trên bản đồ chiến thuật sẽ được quân lính tự động chiếm nếu không có kẻ thù đứng xung quanh công trình đó. Công trình sau khi chiếm có thể dùng để tự động tăng vàng khi qua lượt, phục hồi máu cho các quân lính đứng xung quanh công trình, mua lính mới.

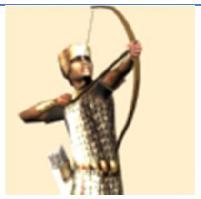
Danh sách công trình			
STT	**Tên công trình**	**Hình đại diện**	**Số vàng phát sinh mỗi lượt**
1	Boat building		50
2	Consulate		60
3	War Academy		70
4	Village		50
5	Bansho		60

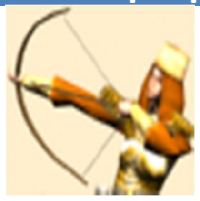
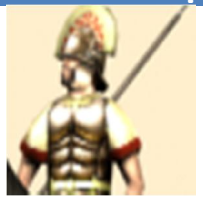
6	Palace		100
7	Factory		120
8	Dojo		150
9	Stable		200
10	Main building		300
11	Top building		80

4.3.4. Quân lính

Người chơi có thể điều khiển quân lính của mình để tấn công phe đối phương. Để mua một loại quân lính sẽ tốn một lượng vàng nhất định. Mỗi quân lính thuộc về một dân tộc trong game. Trong game có 3 loại dân tộc:

Quân lính dân tộc Egypt			
Quân lính	**Hình đại diện**	**Quân lính**	**Hình đại diện**

Elite Archer		Nile spear	
Numidian		Long axe	
Spear cavarly		Egypt General	

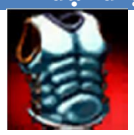
Quân lính dân tộc Greek			
Quân lính	**Hình đại diện**	**Quân lính**	**Hình đại diện**
Amazon Archer		Long spear	
Peltast		Hoplite	
Horse light		Greek general	

Quân lính dân tộc Roman			
Quân lính	**Hình đại diện**	**Quân lính**	**Hình đại diện**

Auxillia archer		Thracian	
Velite		Triarii	
Hastati cavarly		Roman general	
Arcanii			

4.3.5. Kỹ năng

Kỹ năng giúp một loại quân lính có thể phát triển theo nhiều hướng khác nhau. Điều này làm cho sức đánh của các quân lính cùng loại trên chiến trường ít khi giống nhau.

Danh sách kỹ năng			
STT	**Kỹ năng**	**Hình đại diện**	**Tác dụng**
1	Craft Armour		-1 +3
2	Defensive Tactics		+2
3	Derenhalle Blade		+2

4	Fearless		+2 +2
5	Fire Sword		+3 -1
6	Holy Arrow		+2
7	Ralin Boot		+1
8	Strengthen		+5
9	Tactician		+2
10	Telescope		+1
11	Batte Axe		+2 -1
12	Magic Water		+12 -1 -1

4.3.6. Bài phép thuật

Ngoài cách tấn công đối phương bằng vũ khí của chính nhân vật, người chơi còn có thể sử dụng các lá bài phép để tấn công trực tiếp đối phương. Mỗi người chơi lúc ban đầu sẽ có một số lượng bài nhất định. Mỗi lượt người chơi chỉ được rút 1 lá.

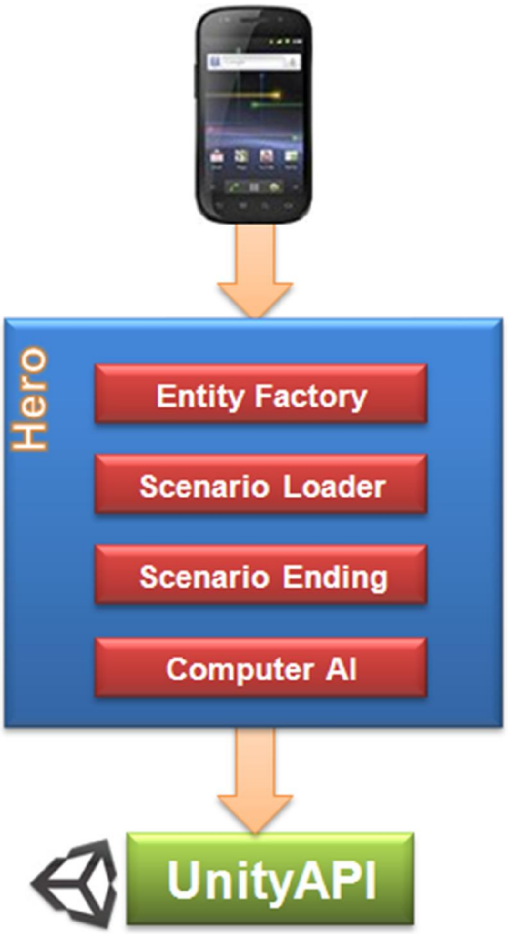
Danh sách bài phép			
STT	**Tên lá bài**	**Hình đại diện**	**Tác dụng**

1	The Judgement Hand		Giảm 5HP của 1 quân lính
2	Goblin Fan		Giảm 7HP của 1 quân lính
3	Horn of the Unicorn		Giảm 5HP tất cả quân lính tại ô tác động và các ô xung quanh ô đó
4	Mystic Plasma Zone		Giảm 9HP của 1 quân lính
5	Paralyzing potion		Giảm 6HP của 1 quân lính
6	Ring of Destruction		Giảm 10HP của 1 quân lính
7	Molten Destruction		Giảm 5HP của tất cả quân lính tại ô tác động và xung quanh ô đó
8	Horn of Heaven		Giảm 9HP của 1 quân lính
9	Piercing Light		Giảm 4HP của 1 quân lính

10	Axe of Despair		Giảm 11HP của 1 quân lính

4.4. Kiến trúc trong game

4.4.1. Kiến trúc tổng thể



Hình 4.6 Kiến trúc tổng thể

Hình 4.6 thể hiện kiến trúc tổng thể của game chúng em xây dựng dựa trên nền tảng Unity. Các thành phần chính trong kiến trúc này bao gồm:

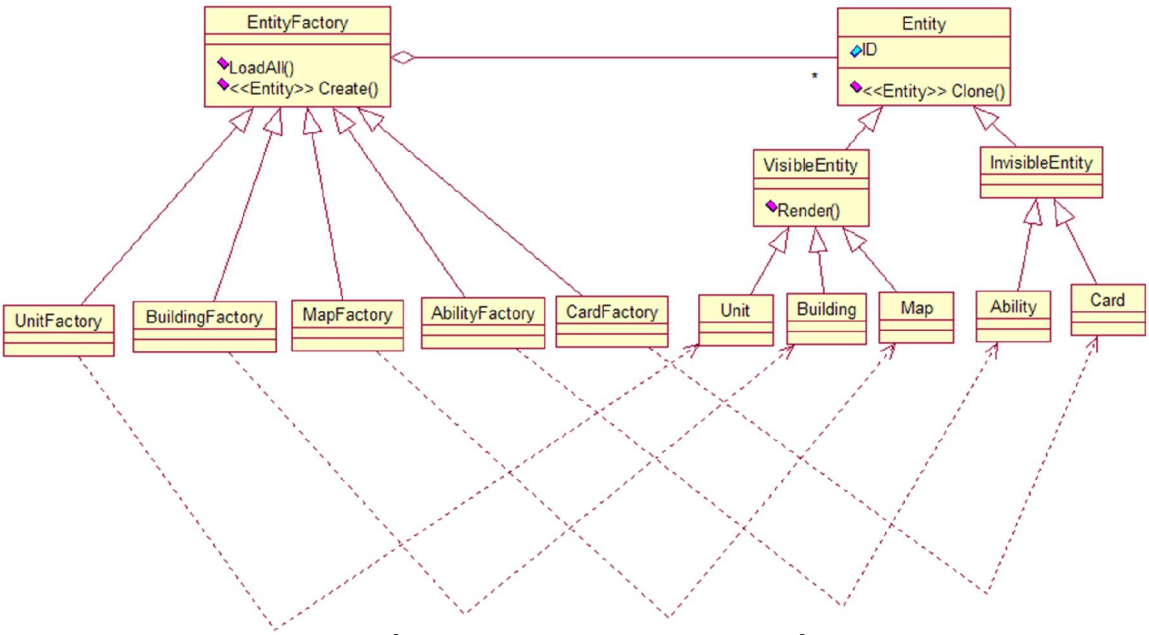
- **Entity Factory**: nhằm phục vụ việc tạo lập các đối tượng trong game. Các đối tượng trong game có thể là các đơn vị lính, công trình, bản đồ, và những tính

năng phép thuật có thêm trong game (xem chi tiết về phân hệ này trong phần 4.4.2)

- **Scenario Loader**: Phân hệ này nhằm giúp nạp game từ file scenario đã được đặc tả trước bên ngoài (xem chi tiết về phân hệ này trong phần 4.4.3)
- **Scenario Ending**: Phân hệ này nhằm đảm bảo game có khả năng dễ dàng thay đổi, bổ sung hay thêm các sự kiện kết thúc màn chơi khác nhau (xem chi tiết về phân hệ này trong phần 4.4.4)
- **Computer AI**: Phân hệ này nhằm đảm bảo game có khả năng dễ dàng thay đổi, bổ sung hay thêm các AI khác nhau (xem chi tiết về phân hệ này trong phần 4.4.5)

4.4.2. Kiến trúc xử lý, phát sinh đối tượng

Trong game, chúng em sử dụng cơ chế quản lý tập trung, tất cả dữ liệu thông tin game chỉ load 1 lần duy nhất lúc khởi tạo game. Với mỗi thực thể trong game sẽ có 1 lớp Factory có nhiệm vụ đọc dữ liệu theo cấu trúc định nghĩa sẵn trong file xml và lưu thông tin vào Entity tương ứng.

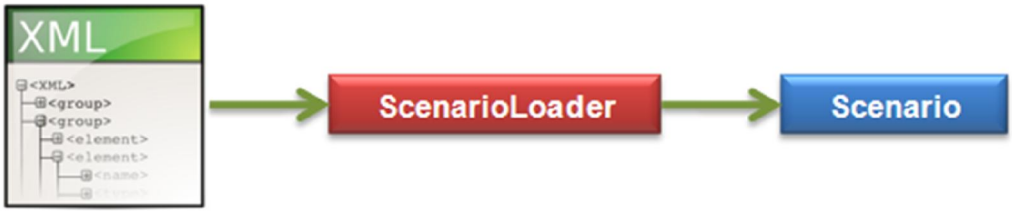


Hình 4.7 Sơ đồ các lớp khởi tạo và phát sinh đối tượng

Việc tạo lập này được xử lý theo kiến trúc được đề nghị ở hình , tương ứng với mỗi loại thực thể cho trước như Unit, Building, Map,… sẽ có một lớp đối tượng chuyên tạo lập các đối tượng thuộc về nhóm tương ứng. Ví dụ như UnitFactory chuyên để xử lý, tạo lập các đối tượng về Unit,... Ở đây, cấu trúc về các thực thể trong game được chia làm 2 nhóm: nhóm Visible là những đối tượng có khả năng hiển thị trên màn hình nên được cung cấp một phương thức là Render() dùng để hiển thị đối tượng, và những đối tượng khác được xử lý về mặt logic trong game nhưng không có nhu cầu hiển thị trên màn hình như Card hay Ability. Tương ứng với mỗi loại thực thể (Entity) có trong game sẽ được quản lý việc tạo lập bởi một lớp đối tượng cụ thể kế thừa từ EntityFactory tương ứng.

Với kiến trúc trên, khi cần tạo một thực thể thật sự, ta chỉ cần gọi hàm Create() với tham số là ID của Entity cần tạo trong lớp Factory tương ứng với loại thực thể.

4.4.3. Kiến trúc nạp màn chơi



Hình 4.8 Load màn chơi từ file xml

Trong một file đặc tả màn chơi cần có những thông tin sau:

- Tên màn chơi
- ID của map (được đặc tả ở file khác)
- Điều kiện kết thúc màn chơi (xem chi tiết hơn ở phần 4.4.4)
- PlayerList: định nghĩa thông tin các người chơi trong game, với quy ước player đầu tiên do người điều khiển, còn lại do máy điều khiển. Thông tin 1 người chơi bao gồm:
  - Tên người chơi
  - Thông tin mô tả ngắn gọn

- Loại AI (người điều khiển thì ta cho bằng 0)
- Số vàng lúc đầu của người chơi
- Màu sắc của người chơi (0→5)
- ID của quốc gia (được đặc tả ở file khác)
- Phe (những người chơi cùng phe thì có giá trị bằng nhau, ngược lại là kẻ thù)
- Danh sách quân lính ban đầu của người chơi, phải luôn có 1 anh hùng (UnitList)
- Danh sách Card của người chơi (CardList)
- Các loại quân lính người chơi có thể mua trong màn (AllowBuyUnitList)

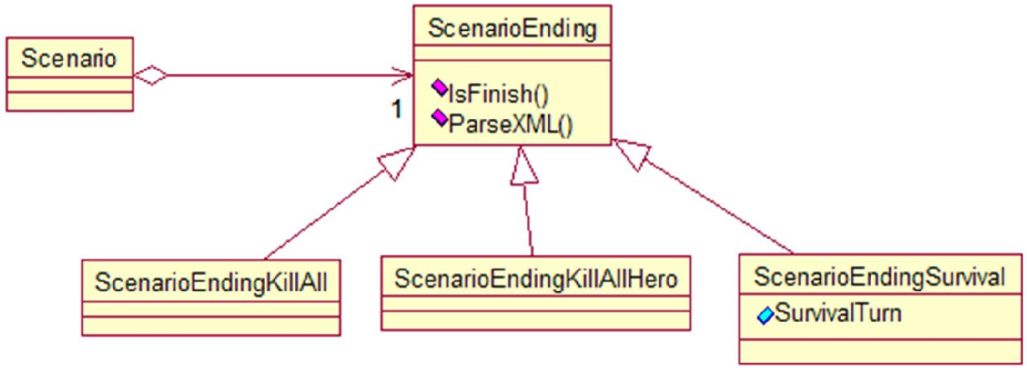
Lớp ScenarioLoader có nhiệm vụ đọc file thông tin màn chơi đã được đặc tả sẵn bên ngoài để tạo ra màn chơi trong game. Với cách xử lý này, ta có thể tạo ra màn chơi mới nhanh chóng và dễ dàng.

```
<?xml version="1.0" encoding="utf-8" ?>
<Scenario>
  <Name> Tên màn chơi </Name>
  <Description> Mô tả màn chơi </Description>
  <MapID> ID của bản đồ trong màn chơi </MapID>
  <EndingCondition type="1">
    <!—Xem chi tiết ở phần 4.4.4 -->
  </EndingCondition>

  <PlayerList n=" Số lượng người chơi ">
    <Player>
      <Name> Tên người chơi </Name>
      <Description> Thông tin ngắn gọn </Description>
      <AI> Loại AI (0: không có AI, 1: hiếu chiến, 2: AI trung hòa) </AI>
      <StartGold> Số vàng lúc đầu chơi</StartGold>
      <ColorIndex> Màu sắc người chơi (0->5) </ColorIndex>
      <NationID> ID của quốc gia </NationID>
      <Side> Phe (Các người chơi cùng phe có giá trị bằng nhau) </Side>
      <UnitList>
        <Unit id=" ID của unit ">
          <Row> Vị trí dòng đặt unit </Row>
          <Col> Vị trí cột đặt unit </Col>
```

```
            <UpgradePoint> Điểm kỹ năng lúc đầu </UpgradePoint>
            <IsHero> Có phải là anh hùng không (1: có, 0: không) </IsHero>
          </Unit>
          <!– các Unit tiếp theo... -->
        </UnitList>
        <CardList>
          <Card id=" ID của Card "></Card>
          <!– các Card tiếp theo... -->
        </CardList>

        <AllowBuyUnitList n=" Số lượng unit ">
          <Unit id=" ID của unit "></Unit>
          <!– các Unit tiếp theo... -->
        </AllowBuyUnitList>

      </Player>
      <!– các Player tiếp theo... -->
    </PlayerList>

</Scenario>
```

Hình 4.9 Nội dung file xml mô tả một màn chơi

4.4.4. Kiến trúc quản lý sự kiện kết thúc màn chơi



Hình 4.10 Sơ đồ lớp quản lý sự kiện kết thúc màn chơi

Mỗi màn chơi trong game sẽ có sự kiện kết thúc màn chơi khác nhau được quản lý bởi các lớp đối tượng xử lý kết thúc màn chơi. Các đối tượng này được cung cấp

hàm IsFinish() có nhiệm vụ kiểm tra khi nào màn chơi kết thúc. Hiện nay game hỗ trợ 3 dạng màn chơi: giết tất cả lính, giết tất cả tướng, sống sót qua mấy lượt. Để làm điều đó, trong game hỗ trợ lưu đặc tả sự kiện kết thúc màn chơi bên trong file đặc tả màn chơi bằng thẻ EndingCondition với cấu trúc sau:

```
// File đặc tả màn chơi Scenario.xml
//...
<EndingCondition type="1">
    <!– Thông tin của sự kiện kết thúc màn chơi -->
</EndingCondition>
//...
```

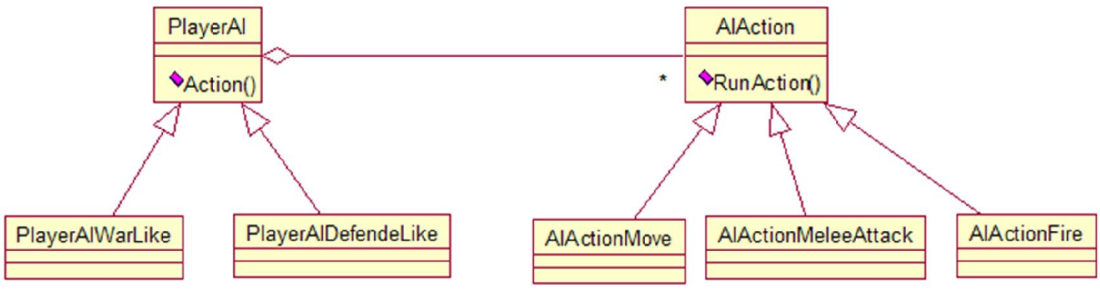
Thuộc tính type qui định loại sự kiện kết thúc màn chơi:

- type =1: kết thúc màn chơi khi giết tất cả lính
- type =2: kết thúc màn chơi khi giết tất cả tướng
- type =3: kết thúc màn chơi khi sống sót qua bao nhiêu lượt

Tùy vào type mà ta sẽ tạo lập ra đối tượng cụ thể và dùng hàm ParseXML() để đối tượng tự đọc nội dung bên trong thẻ EndingCondition lấy thông tin xử lý kết thúc màn chơi.

Với kiến trúc theo mẫu Strategy trên ta có thể thêm sự kiện kết thúc màn chơi mới vào game dễ dàng.

4.4.5. Kiến trúc quản lý AI



Hình 4.11 Sơ đồ lớp quản lý AI trong game

Trong game, máy có nhiều dạng AI khác nhau. Mỗi AI được quản lý bởi một lớp đối tượng kế thừa từ lớp PlayerAI. Trong các lớp này được cung cấp hàm Action()

để tính toán bước đi của máy và trả ra một danh sách AIAction. Máy có nhiều hành động khác nhau như đi, đánh, bắn, … Tương ứng với mỗi hành động sẽ được thực thi trong hàm RunAction() của một lớp đối tượng kế thừa từ AIAction.

Chương 5

Kết luận và hướng phát triển

✍ *Nội dung của chương này trình bày các kết quả đạt được trong quá trình thực hiện luận văn và hướng phát triển của đề tài.*

5.1. Các kết quả đạt được

5.1.1. Ứng dụng game

Sau khi tìm hiểu các phương pháp xây dựng game 3D với engine Unity cho Android chúng em đã hoàn thành ứng dụng game thuộc thể loại chơi theo lượt. Trong quá trình xây dựng game, chúng em đã nêu ra một số vấn đề gặp phải, và đưa ra một số hướng giải pháp để giải quyết.

Các màn hình giao diện chính của game mà chúng em đã xây dựng trong luận văn này:

Hình 5.1 Màn hình menu chính của Game

Ở màn hình Menu chính, người chơi có thể chọn “Play Campaign” để chọn màn chơi, màn đang chơi hoặc chơi qua rồi sẽ có màu xanh, màn chưa chơi sẽ có màu tối và chưa được quyền chơi.

Hình 5.2 Màn hình menu chọn màn chơi.

Hình 5.3 Một cảnh khi chọn vào quân lính.

Hình 5.4 Một cảnh đánh nhau giữa quân lính 2 phe.

Hình 5.5 Một cảnh phóng lao của lính.

Hình 5.6 Một cảnh chọn vào nhà đã chiếm được để mua lính

Hình 5.7 Màn hình nâng cấp kỹ năng

Hình 5.8 Màn hình khi chọn xem thông tin lá bài.

Hình 5.9 Một cảnh khi kéo lá bài phép vào lính.

5.2. Hướng phát triển

- Tối ưu hóa các xử lý giúp game chạy nhanh hơn.
- Game sẽ hỗ trợ chế độ nhiều người chơi cùng một lúc qua Bluetooth hoặc Internet.
- Tăng thêm số lượng màn chơi để người chơi thêm nhiều lựa chọn.
- Cung cấp thêm các công cụ để người chơi có thể chỉnh sửa hoặc tạo ra cho mình riêng các hình ảnh nhân vật, hiệu ứng âm thanh, …
- Cải thiện trí thông minh nhân tạo của máy.

Danh mục tài liệu tham khảo

[1] Ryan Henson Creighton, *Unity 3D Game Development by Example*, Packt Publishing, 2010.

[2] Will Goldstone, *Unity Game Development Essential*, Packt Publishing, 2009.

[3] Efraim Meulenberg, *Game Development with Unity*, VTC, 2010.

[4] Efraim Meulenberg, *Game Development for iPhone/iPad Using Unity iPhone*, VTC, 2010.

[5] TornadoTwins**,** *Wormgame Cartoony Series* http://www.unityprefabs.com/wormgame-cartoony-series-tutorial.html

[6] Zak Parrish, *Unity Fundamentals* http://www.3dbuzz.com/vbforum/content.php?176

[7] Chad and Eric, *Unity Training* http://walkerboystudio.com/html/unity_training___free__.html