**ĐẠI HỌC DUY TÂN**

**KHOA CÔNG NGHỆ THÔNG TIN**

**--- oOo ---**

**ĐỒ ÁN TỐT NGHIỆP**

**NGÀNH CÔNG NGHỆ THÔNG TIN – KHÓA K19**

**ĐỀ TÀI:**

***XÂY DỰNG ỨNG DỤNG GAME 2D ĐA NỀN TẢNG VỚI COCOS2D-X GAME ENGINE***

**Giảng viên hướng dẫn: ThS. Võ Minh Thông**

**Sinh Viên thực hiện: Nguyễn Chiêm Minh Vũ**

**Lớp: K19 TPM1**

**Niên khoá: 2014 – 2017**

**Đà Nẵng, tháng 03/2017**

**LỜI NÓI ĐẦU**

Trong những năm gần đây, game mobile đã trở thành một phần không thể thiếu của ngành công nghiệp game toàn cầu và đang không ngừng mở rộng một cách nhanh chóng. Góp phần vào sự phát triển mạnh mẽ của ngành công nghiệp game không chỉ có những công ty, tập đoàn phát triển game lớn mà còn có sự tham gia của rất nhiều những nhà phát triển game cá nhân. Với các công cụ phát triển game đa dạng hiện có, không quá khó để những nhà phát triển game độc lập tự xây dựng những ứng dụng game cho riêng mình cho dù là khởi đầu từ con số không.

Với mong muốn góp phần vào sự phát triển của ngành công nghiệp giải trí ảo, em đã chọn đề tài "Xây dựng ứng dụng game 2D đa nền tảng" nhằm tích lũy thêm kinh nghiệm, đồng thời cũng cho ra mắt một sản phẩm ứng dụng game đủ hoàn thiện để người chơi có thể trải nghiệm.

Đồ án sẽ tập trung nghiên cứu về Cocos2d-x Game Engine - một game engine gọn nhẹ nhưng không kém phần mạnh mẽ so với các game engine lớn hiện có, nhằm thực nghiệm phát triển một game 2D hoàn chỉnh. Các chương đầu trong tài liệu sẽ giới thiệu tổng quan về ngành công nghiệp phát triển game, xu hướng phát triển game trên thế giới, các công nghệ được sử dụng trong các công ty phát triển game lớn, tìm hiểu về Cocos2d-x game engine. Các chương tiếp theo sẽ đi vào phân tích và thiết kế game 2D mang tên "Beware of dungeon".

**LỜI CẢM ƠN**

Trong thời gian làm đồ án tốt nghiệp, em đã nhận được nhiều sự giúp đỡ, đóng góp ý kiến từ phía thầy cô, gia đình và bạn bè.

Em xin gửi lời cảm ơn chân thành đến Th.s Võ Minh Thông, giảng viên khoa Điện - Điện tử, đã trực tiếp hướng dẫn trong suốt quá trình thực hiện khóa luận.

Em cũng xin chân thành cảm ơn Th.s Nguyễn Minh Nhật và Th.s Lê Thị Ngọc Vân cùng các giảng viên khoa Công Nghệ Thông Tin trường Đại học Duy Tân Đà Nẵng đã luôn tận tình chỉ bảo trong suốt những năm tháng học Đại học để em có được kiến thức nền tảng vững vàng, sẵn sàng cho công việc trong tương lai.

Em đặc biệt cảm ơn freelancer artist Maria Kalyagina đến từ Nga đã tham gia vào dự án để góp phần hoàn thiện sản phẩm.

Cuối cùng, em xin cảm ơn gia đình và bạn bè đã luôn ủng hộ, giúp đỡ em trong suốt quá trình học tập và hoàn thành đồ án tốt nghiệp.

Với điều kiện về thời gian cũng như kinh nghiệm hạn chế, đồ án này không thể tránh khỏi những thiếu sót. Em rất mong nhận được thêm sự chỉ bảo, đóng góp ý kiến để sản phẩm được hoàn thiện tốt nhất có thể.

Em xin chân thành cảm ơn!

**Đà Nẵng, 13 tháng 04 năm 2017**

**MỤC LỤC**

[Chương 1 TỒNG QUAN 1](#_Toc481101909)

[1.1 Tên đề tài 1](#_Toc481101910)

[1.2 Tính cấp thiết của đề tài 1](#_Toc481101911)

[1.3 Mục đích nghiên cứu 4](#_Toc481101912)

[1.4 Đối tượng và phạm vi nghiên cứu 4](#_Toc481101913)

[1.5 Phương pháp nghiên cứu 5](#_Toc481101914)

[1.6 Ý nghĩa khoa học và thực tiễn của đề tài 6](#_Toc481101915)

[1.7 Kết cấu của đồ án 6](#_Toc481101916)

[Chương 2 TÌM HIỂU NGÀNH CÔNG NGHIỆP PHÁT TRIỂN GAME 7](#_Toc481101917)

[2.1 Tổng quan ngành công nghiệp phát triển game 7](#_Toc481101918)

[2.1.1 Ngành công nghiệp phát triển game 7](#_Toc481101919)

[2.1.2 Quy trình phát triển game 8](#_Toc481101920)

[2.2 Các xu hướng phát triển game trên thế giới 10](#_Toc481101921)

[2.2.1 Các xu hướng đang được hướng đến 10](#_Toc481101922)

[2.2.2 Một số xu hướng nên thay đổi 16](#_Toc481101923)

[2.3 Các công nghệ phát triển game đang được sử dụng phổ biến 20](#_Toc481101925)

[2.4 Tổng quan về game engine 26](#_Toc481101926)

[Chương 3 TÌM HIỂU VỀ COCOS2D-X GAME ENGINE 28](#_Toc481101927)

[3.1 Cocos2d-x là gì? 28](#_Toc481101928)

[3.2 Những thành phần cơ bản của Cocos2d-x 29](#_Toc481101929)

[3.3 Sprites 42](#_Toc481101930)

[3.3.1 Sprite là gì? 42](#_Toc481101931)

[3.3.2 Tạo đối tượng Sprite với Cocos2d-x 42](#_Toc481101932)

[3.3.3 Tạo đối tượng Sprite từ Sprite Sheet 42](#_Toc481101933)

[3.3.4 Công cụ tạo ra các Sprite Sheet 43](#_Toc481101934)

[3.3.5 Thao tác với Sprite 44](#_Toc481101935)

[3.4 Actions 46](#_Toc481101936)

[3.4.1 Phân biệt Action By và Action To 47](#_Toc481101937)

[3.4.2 Chạy các Action đơn giản 48](#_Toc481101938)

[3.5 Scenes 55](#_Toc481101939)

[3.5.1 Scene là gì? 55](#_Toc481101940)

[3.5.2 Tạo ra một Scene với Cocos2d-x 55](#_Toc481101941)

[3.5.3 Scene Transition 56](#_Toc481101942)

[3.6 Các thành phần UI 59](#_Toc481101943)

[3.6.1 Widgets 59](#_Toc481101944)

[3.6.2 Labels 59](#_Toc481101945)

[3.6.3 Menu và các Menu Item 61](#_Toc481101946)

[3.6.4 Buttons 63](#_Toc481101947)

[3.6.5 Checkboxs 64](#_Toc481101948)

[3.6.6 TextField 65](#_Toc481101949)

[3.7 Một số kiểu Node khác 67](#_Toc481101950)

[3.7.1 Tile Map 67](#_Toc481101951)

[3.7.2 Particle System 69](#_Toc481101952)

[3.7.3 Parallax 71](#_Toc481101953)

[3.8 Điều phối sự kiện 72](#_Toc481101954)

[3.8.1 Cơ chế hoạt động của đối tượng điều phối sự kiện 72](#_Toc481101955)

[3.8.2 Các kiểu của đối tượng lắng nghe sự kiện 73](#_Toc481101956)

[3.8.3 Độ ưu tiên của sự kiện 73](#_Toc481101957)

[3.8.4 Xử lý các kiểu sự kiện 74](#_Toc481101958)

[3.8.5 Đăng kí và xóa bỏ đối tượng xử lý sự kiện 78](#_Toc481101959)

[3.9 Audio 79](#_Toc481101960)

[3.10 Game controller 82](#_Toc481101961)

[3.11 Physics 87](#_Toc481101962)

[3.11.1 Physics engine - có cần thiết hay không? 87](#_Toc481101963)

[3.11.2 Những thuật ngữ và khái niệm thường dùng 88](#_Toc481101964)

[3.11.3 Physics world và Physics body 90](#_Toc481101965)

[3.11.4 Collision 91](#_Toc481101966)

[3.11.5 Collision detection 92](#_Toc481101967)

[Chương 4 PHÂN TÍCH, THIẾT KẾ GAME BEWARE OF DUNGEON 95](#_Toc481101968)

[4.1 Giới thiệu game 95](#_Toc481101969)

[4.1.1 Cốt truyện của game 95](#_Toc481101970)

[4.1.2 Các quy luật trong game 95](#_Toc481101971)

[4.2 Kiến trúc của game 97](#_Toc481101972)

[4.3 Các tính năng của game 98](#_Toc481101973)

[4.3.1 Các tính năng chính 98](#_Toc481101974)

[4.3.2 Một số tính năng mở rộng 99](#_Toc481101975)

[4.4 Phân tích, thiết kế game 100](#_Toc481101976)

[4.4.1 Hệ thống chuyển đổi Scene 100](#_Toc481101977)

[4.4.2 Hệ thống bản đồ, màn chơi 102](#_Toc481101978)

[4.4.3 Hệ thống phát sinh quái vật cho mỗi màn chơi 102](#_Toc481101979)

[4.4.4 Hệ thống chia sẻ 102](#_Toc481101980)

[Chương 5 TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ 103](#_Toc481101981)

[DANH MỤC TÀI LIỆU THAM KHẢO 104](#_Toc481101982)

**DANH MỤC CÁC TỪ VIẾT TẮT**

**DANH MỤC BẢNG BIỂU, SƠ ĐỒ**

**DANH MỤC HÌNH ẢNH**

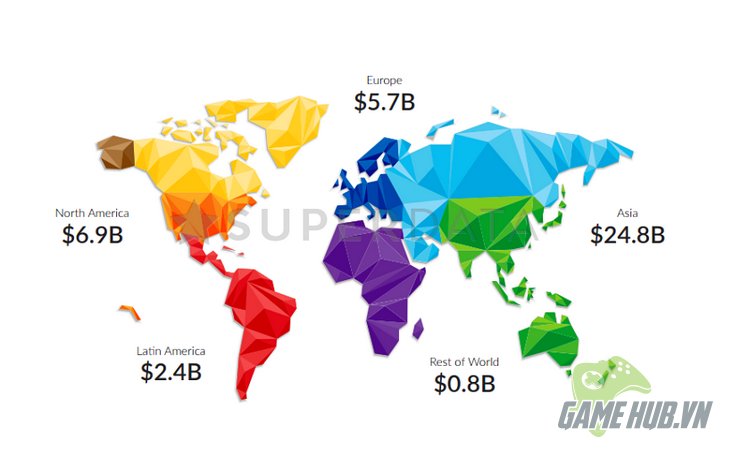
# TỒNG QUAN

## Tên đề tài

Xây dựng ứng dụng game 2D đa nền tảng.

## Tính cấp thiết của đề tài

Trong những năm gần đây, ngành công nghiệp Game đã phát triển với tốc độ rất nhanh. Báo cáo về mảng Game mobile trong năm 2016 đã cho thấy giá trị khổng lồ mà ngành công nghiệp giải trí ảo trên điện thoại di động đã mang lại.



Báo cáo về mảng Game mobile trong năm 2016 đã cho thấy mức doanh thu của thị trường này đạt mức kỷ lục 40 tỷ USD, tương đương 906 ngàn tỷ VND. Doanh thu của thị trường Game mobile 2016 đã đạt mức tăng trưởng 18% so với năm 2015. Bản báo cáo được thực hiện bởi công ty nghiên cứu thị trường SuperData Research.

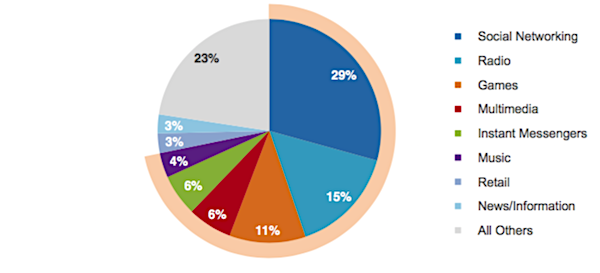
Về khía cạnh khu vực, chúng ta có thể thấy Châu Á vẫn là thị trường phát triển game mạnh nhất so với tất cả các khu vực khác trên thế giới khi đạt đến doanh thu 24,8 tỷ USD trong năm 2016, vượt xa khu vực Bắc Mỹ và Châu Âu.

Với việc cộng đồng ngày càng dành nhiều thời gian hơn với Game mobile, chúng ta có thể thấy được cơ hội của các nhà phát triển game từ doanh nghiệp đến cá nhân đang dần được mở rộng nhờ sự phổ biến của hình thức giải trí ảo trên điện thoại di động.

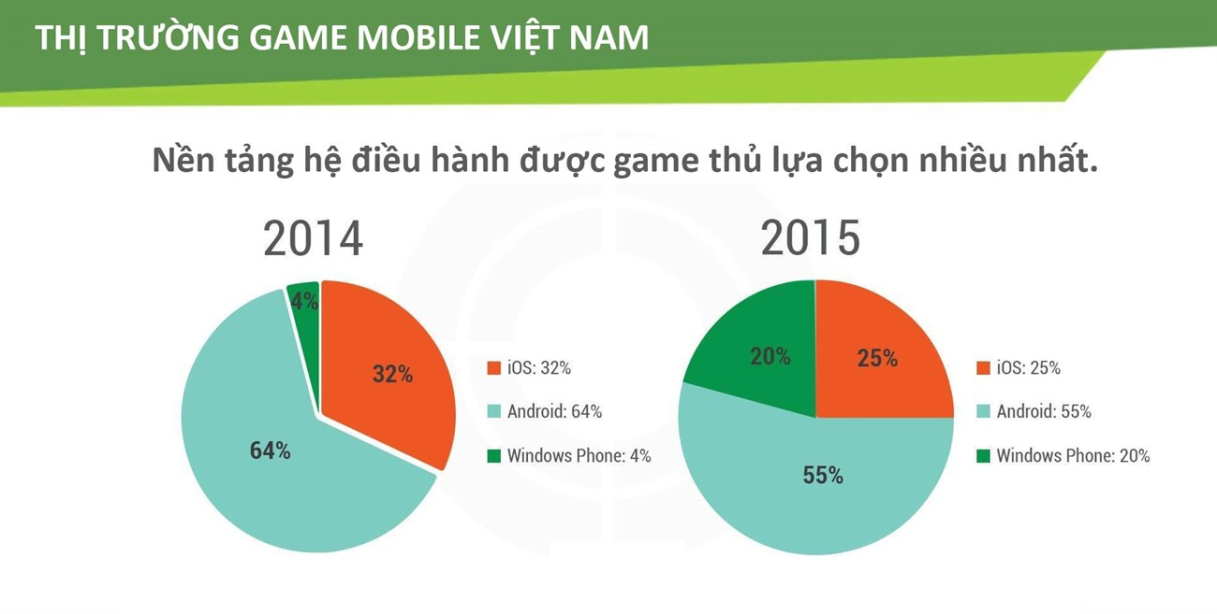


Theo báo cáo của công ty công nghệ tiên phong trong việc cung cấp các nền tảng tiện ích cho điện thoại thông minh APPOTA, vào đầu năm 2016, số lượng người dùng điện thoại thông minh đã đạt con số 1,9 tỷ người - chiếm khoảng 25% dân số thế giới và 42% số lượng người sử dụng điện thoại di động. Trong số đó, số lượng người dùng hệ điều hành Android chiếm đến 82%.

Trở lại với thị trường Game mobile Việt Nam, trong năm 2016, thị trường game Việt Nam phát triển mạnh mẽ với hơn 200 đầu game được ra mắt. Trong đó, chỉ riêng game mobile đã chiếm đến gần 75% với tổng số gần 150 sản phẩm được tung ra thị trường. Với sự tăng trưởng nhanh của ngành công nghiệp smart phone, kết hợp với sự tăng trưởng kinh tế của mặt bằng chung trong xã hội, việc sở hữu smart phone hiện nay không hề khó, do đó, nhu cầu giải trí trên hệ smart phone dần phổ biến vì tính tiện dụng, dễ dàng giải trí mọi lúc mọi nơi.



Biểu đồ nghiên cứu hành vi sử dụng điện thoại di động cho thấy mức độ sử dụng của các loại ứng dụng trên smart phone trong tổng số thời gian sử dụng smart phone, trong đó, các ứng dụng Games vẫn nằm trong top 3 các loại ứng dụng được sử dụng nhiều nhất trên smart phone.



Đây là biểu đồ nhằm so sánh tỉ lệ hệ điều hành được game thủ việt nam chọn sử dụng trên smart phone để chơi game. Kết quả cho thấy hệ điều hành Android vẫn đang chiếm tổng số lớn nhất trong hầu hết các loại hệ điều hành smart phone hiện nay.

Bên cạnh đó, chúng ta còn cần xem xét đến những lợi ích mà Game đem lại. Việc chơi Game trước tiên là để giải trí sau một quãng thời gian làm việc, học tập mệt mỏi. Game cũng có thể mang lại cho chúng ta nhiều kiến thức bổ ích trong cuộc sống thông qua các thể loại game nhập vai, mô phỏng thế giới thực, hoặc một số game thuộc thể loại trí tuệ cũng giúp chúng ta rèn luyện kỹ năng phân tích, giải quyết vấn đề, kích thích tính sáng tạo, rèn luyện tiếng Anh, …

Ngày nay, việc tạo ra một game đơn giản trong một khoảng thời gian ngắn không còn là một công việc quá khó khăn nhờ sự phát triển của rất nhiều bộ công cụ hỗ trợ, và hơn thế nữa là có rất nhiều Game engine miễn phí để chúng ta lựa chọn cho việc xây dựng một sản phẩm Game từ con số không. Chúng ta có thể liệt kê ra một số Game engine phổ biến như Unity, Game Maker, CryEngine, LibGDX, Cocos2d-x,… Đồ án sẽ tập trung nghiên cứu phương pháp phát triển Game 2D trên nền tảng hệ điều hành Android sử dụng Cocos2d-x engine. Đây là một engine mã nguồn mở có kích thước khá gọn nhẹ, được sử dụng để phát triển game trên nhiều nền tảng hệ điều hành khác nhau.

## Mục đích nghiên cứu

Đồ án được thực hiện với mục đích nghiên cứu, xây dựng một ứng dụng giải trí trên nền tảng điện thoại di động chạy hệ điều hành Android sử dụng Cocos2d-x engine.

## Đối tượng và phạm vi nghiên cứu

Tập trung chính vào đối tượng nghiên cứu là sử dụng Cocos2d-x engine trong quá trình phát triển ứng dụng game 2D, trên cơ sở:

* Tìm hiểu về game engine và Cocos2d-x engine.
* Tìm hiểu về một số công cụ cần thiết hỗ trợ cho quá trình phát triển game.
* Xác định các vấn đề và đưa ra các giải pháp trong quá trình xây dựng game trên hệ điều hành Android với Cocos2d-x engine.
* Sau đó, đồ án sẽ đi vào thiết kế và xây dựng ứng dụng game. Cuối cùng sẽ trình bày về các kết quả đã đạt được, những tồn tại và hướng phát triển của đề tài.
* Đồ án chỉ dừng lại ở phạm vi phát hành ra một ứng dụng game 2D offline với những yêu cầu đặt ra như: xây dựng được một game trí tuệ có cách chơi đơn giản nhưng khó để vượt qua thử thách, game có thể chơi được ở mọi lúc mọi nơi, game phù hợp với nhiều lứa tuổi khác nhau…

## Phương pháp nghiên cứu

* Tìm hiểu cách sử dụng engine thông qua tài liệu được cung cấp ngay trên trang chủ của Cocos2d-x engine (<http://www.cocos2d-x.org/>).
* Xây dựng thử nghiệm một số phiên bản clone từ những game đơn giản.

## Ý nghĩa khoa học và thực tiễn của đề tài

Phát triển Game trên điện thoại di động đã trở thành một xu hướng dành được nhiều sự quan tâm của phần lớn các bạn trẻ Việt Nam và trên thế giới. Thực hiện đề tài này sẽ giúp tiếp cận, nghiên cứu, hoàn thiện hơn về kỹ năng cũng như quy trình phát triển Game.

Trong tương lai, khi mà những chiếc điện thoại thông minh hoàn toàn thay thế các dòng máy tính để bàn hay máy tính xách tay, ngành công nghiệp phát triển Game mobile cũng sẽ khẳng định một vị trí quan trọng trong lĩnh vực giải trí.

## Kết cấu của đồ án

* Tìm hiểu ngành công nghiệp phát triển game.
* Tìm hiểu về Cocos2d-x game engine.
* Phân tích và thiết kế game Beware of Dungeon.
* Triển khai ứng dụng, đánh giá kết quả.
* Kết luận và hướng phát triển.

# TÌM HIỂU NGÀNH CÔNG NGHIỆP PHÁT TRIỂN GAME

## Tổng quan ngành công nghiệp phát triển game

### Ngành công nghiệp phát triển game

Ngành công nghiệp phát triển game là một lĩnh vực hoạt động kinh tế liên quan đến sự phát triển thương mại của những video game. Nó tạo ra việc làm cho hàng ngàn người trên thế giới.

Ngành công nghiệp máy tính và công nghiệp game đã trở thành tiêu điểm trong thị trường, nó thu về khoảng 9.5 tỷ USD trong năm 2007, 11.7 tỷ USD trong năm 2008, và lên đến 25.1 tỷ USD trong năm 2010 theo báo cáo của ESA.

Sự phát triển của ngành công nghiệp game đã thúc đẩy sự phát triển của các dòng PC hiện đại: card âm thanh, card đồ họa, tăng tốc CPU, 3D graphics libraries, và các tiến trình xử lý đồng bộ được cải tiến. Card đồ họa được thiết kế tích hợp thêm nhiều màu sắc, các thuật toán xử lý 3D được tối ưu hóa, …

Trước đây, chi phí cho việc phát triển game khá nhỏ, nên games được phát triển bởi những cá nhân, hoặc một team nhỏ bao gồm vài lập trình viên và họa sĩ. Nhiều game chỉ tốn khoảng một vài tháng để triển khai, vì thế những nhà phát triển có thể cho ra mắt nhiều game trong một năm. Nhưng với sự phát triển của công nghệ máy tính và đồ họa, nó đòi hỏi số lượng nhà phát triển trong team tăng lên để phân tích và thiết kế những game phức tạp hơn. Một team lớn hơn bao gồm những lập trình viên (programmers), những họa sĩ (artists), những nhà thiết kế game (game designers), và những nhà sản xuất game (game producers).

Ngày nay, những video game có kích thước lớn với những chi tiết kỹ thuật phức tạp mang về doanh thu vô cùng lớn, gây tác động lớn đến nền kinh tế thị trường. Ví dụ game "Call of Duty: Black Ops" mang về lợi nhuận hơn 650 tỷ USD chỉ trong 5 ngày ra mắt đầu tiên.

### Quy trình phát triển game

Phát triển game là một quy trình của việc tạo ra các video game, được thực hiện bởi các nhà phát triển game (có phạm vi từ một cá nhân cho tới một tập đoàn lớn). Các game thương mại truyền thống chơi trên PC hay console thường được tài trợ bởi một nhà xuất bản và mất vài năm để phát triển, trong khi một số video game được phát triển độc lập được xuất bản trong thời gian ngắn hơn. Những game được phát triển độc lập đang nổi lên trong những năm gần đây, góp phần làm đa dạng thế giới game và thị trường game.

Tham gia vào quy trình phát triển game có những vai trò như sau:

* Producer

Nếu một game có tầm cở quy mô lớn, nó thường được giám sát bởi 1 hoặc nhiều game producer với mục đích quản lý team phát triển, lập kế hoạch, báo cáo tiến độ, quản lý ngân sách, … Ngoài ra, game producer cũng có thể tham gia vào việc quản lý chất lượng quy trình, quản lý quá trình beta test, nội địa hóa sản phẩm game. Vai trò này được coi như project manager, project lead hoặc director.

* Publisher

Publisher (nhà xuất bản game) thường là một công ty xuất bản những video game mà họ có, những video game này có thể được phát triển bên trong công ty hoặc bên ngoài công ty. Họ chịu trách nhiệm về sản xuất, tiếp thị sản phẩm game, bao gồm cả nghiên cứu thị trường và quảng cáo. Họ thường tài trợ cho công việc phát triển game, chi trả cho những bộ phận phát triển game phía bên ngoài (thường được gọi là studio).

Một số chức năng khác mà publishers thường đảm nhận là đưa ra quyết định cho việc chi trả cho bản quyền game, chi trả cho việc nội địa hóa sản phẩm game, …

Những nhà phát triển game độc lập thường không có vai trò publisher trong quy trình và họ có những cách khác để phân phối sản phẩm game của họ.

* Development team

Đội ngũ phát triển game thường có phạm vi từ một vài nhóm nhỏ cho đến một studio vài trăm nhân viên chịu trách nhiệm sản xuất vài tựa đề game lớn. Một số nhà phát triển game độc lập thường phải đảm nhận cùng lúc nhiều vai trò khác nhau trong đội ngũ phát triển, nhưng tất cả vai trò trong các phạm vi khác nhau đều có chức năng giống nhau. Một đội ngũ phát triển game bao gồm những vai trò dưới đây:

- Game designer:

Game designer là người thiết kế ra gameplay, đưa ra ý tưởng game, các luật chơi và cấu trúc của một game. Họ là những người có tầm nhìn chiếc lược của một game. Game designer thường được coi là tác giả của game. Trong một game quy mô lớn, thường có những game designer phụ trách những phần khác nhau trong game như thiết kế cơ chế game, giao diện người dùng, thiết kế nhân vật trong game, …

- Artist (Game art design):

Artists là những người tạo ra các tác phẩm nghệ thuật cho game, họ có vai trò quan trọng giúp cho game thu hút được người chơi bằng những tác phẩm đồ họa 2D/3D mà họ vẽ ra.

- Level designer:

Khác với Game designer, level designer là người tạo ra các màn chơi trong game, đặt ra những thử thách hoặc nhiệm vụ cho người chơi. Họ chịu trách nhiệm tạo ra sự cân bằng cho game.

- Programmer:

Game programmer là những người cốt lõi trong quá trình phát triển game. Thường có một hoặc vài người được gọi là lead programmer, phụ trách phát triển phần code cơ sở của các tính năng trong game, sau đó phân chia các module riêng lẻ đến programmer team. Trong việc phát triển game thường có những module chính sau:

* Physics - một trong những phần quan trọng nhất của game, bao gồm kiếm tra va chạm, giả lập môi trường vật lý, …
* Graphics - xử lý đồ họa trong game.
* Sound - tích hợp nhạc, hiệu ứng âm thanh vào game.
* Gameplay - thực hiện các quy luật, tính năng trong game.
* UI - liên quan đến giao diện người dùng (Menu, HUD, help, feedback, …)
* Input processing - xử lý sự kiện người dùng.
* Network communications - một thành phần không thể thiếu trong game multiplayer.

Đây là những thành phần không thể thiếu trong game.

- Game tester:

Là những người làm việc trong bộ phận quản lý chất lượng, họ đảm bảo rẳng sản phẩm game được phát triển đúng với tài liệu thiết kế và mang lại trải nghiệm tốt cho người chơi.

## Các xu hướng phát triển game trên thế giới

### Các xu hướng đang được hướng đến

* Đa nền tảng

Khi thị trường game PC trở nên bão hòa, chúng ta có thể thấy game sở hữu công nghệ đa nền tảng (cross-platform) đang trở thành xu hướng lựa chọn của các nhà phát triển game trên thế giới.

Điểm mạnh của game đa nền tảng chính là người chơi có thể tham gia vào game trên bất kỳ thiết bị nào mà không cần quan tâm đến sự tương thích của game với hệ điều hành đang chạy. Trong giai đoạn phát triển kinh tế như hiện nay, việc một người sở hữu nhiều hơn 1 thiết bị công nghệ (máy tính, laptop, điện thoại, …) là điều rất bình thường, và người chơi cũng mong muốn rằng họ có thể chơi game bằng bất kỳ thiết bị nào mà họ đang cầm trên tay, bởi sự tiện nghi và sự không ràng buộc giúp con người cảm thấy thoải mái hơn.

Nắm bắt được xu thế này, game Candy Crush Saga được tung ra thị trường vào năm 2013 đã trở thành một hiện tượng đáng chú ý khi game hổ trợ chơi game web và thiết bị mobile.



*Candy Crush Saga đã trở thành hiện tượng đáng chú ý của game đa nền tảng trong năm 2013*

* Game bom tấn, console được đưa lên mobile

Từ trước đến nay, các game thủ vẫn luôn mặc định rằng chơi game trên PC là tốt nhất. Chúng ta có thể bắt gặp hình ảnh các game thủ dành hơn 8 giờ trong một ngày để ngồi trước màn hình PC bởi những trãi nghiệm mà nó mang lại. Điều làm nên thành công của game PC có lẽ phần lớn nhờ vào khả năng xử lý đồ họa mạnh mẽ và liên tục trong nhiều giờ, bên cạnh đó là khả năng kết nối Internet ổn định giúp người dùng trải nghiệm game online tốt hơn.

Nhưng hiện nay, số lượng người chơi game mobile đã tăng trưởng một cách vượt trội, và hoàn toàn có thể cạnh tranh với thị trường game PC truyền thống. Điều này cũng dễ hiểu bởi các dòng smart phone hiện nay đều được trang bị cấu hình cao và khả năng xử lý đồ họa mượt mà, cùng với sự phát triển của cơ sở hạ tầng Internet giúp người chơi có thể tham gia trò chơi mọi lúc, mọi nơi.



*Game đua xe Asphalt 8 được phát triển bởi Gameloft với chất lượng đồ họa cao trên nền tảng smart phone*

Với sự tiện nghi mà smart phone đem lại, một số lượng không nhỏ những game thủ đã từ bỏ việc ngồi hàng giờ sang trải nghiệm những tựa game đồ họa cao trên smart phone. Nắm được xu hướng này, ngày càng có nhiều nhà phát triển game tập trung vào mảng game mobile, đồng thời mang những tựa game bom tấn PC sang mobile. Một minh chứng rõ ràng nhất đó chính là game "Liên Quân Mobile", một tựa game được chuyển thể từ "League of Legends" nổi tiếng là game có hệ thống PvP đa dạng nhất trên PC được Riot phát triển.

* Game live stream

Live stream đơn giản là truyền video trực tiếp, nó đang là xu hướng được sử dụng rộng rãi trên Internet, đây là một tính năng được khá nhiều mạng xã hội áp dụng.

Game live stream đang trở thành xu hướng của game thủ trên toàn cầu. Việc theo dõi trực quan các trận đấu chuyên nghiệp là cách tốt nhất để các game thủ học hỏi các kỹ thuật từ những game thủ nổi tiếng khác. Chính vì những điều hữu ích mà một kênh stream mang lại, việc tích hợp stream trực tiếp vào game đã mang đến một luồng gió mới cho game thủ trên thế giới. Nhờ vào game live stream, game thủ nay cũng đã có thể tự kiếm được nguồn thu nhập cho riêng mình chỉ từ việc chơi game trực tuyến.



*League of Legends - một trong những game được tích hợp hệ thống live stream thành công nhất trên thế giới*

* Thực tế ảo

Năm 2016 được mệnh danh là năm của Virtual Reality và năm nay chúng ta sẽ được tiếp tục chứng kiến những thay đổi đáng kể trong trào lưu công nghệ thực tế ảo. Tuy nhiên, do mới được phát triển chưa lâu, những chuyển biến tiếp theo của Virtual Reality sẽ bị phụ thuộc khá nhiều vào mức giá và các nội dung hỗ trợ.

Hiện nay, ngày càng có nhiều hãng công nghệ tham gia vào lĩnh vực phát triển Virtual Reality như Google, Facebook, Samsung, … và trong lĩnh vực game Virtual Reality có sự tham gia của các tập đoàn lớn như Epic Games corporation, Electronic Arts, …



*Game chiến đấu trên Virtual Reality*

Virtual Reality mang lại rất nhiều lợi ích trong thực tế: giả lập môi trường bay, mô phỏng tình huống thực tế, … Và khi đưa game lên Virtual Reality cũng có thể giúp chúng ta học hỏi được nhiều thứ hơn, có nhiều trải nghiệm tuyệt vời hơn. Game trên Virtual Reality chắc chắn sẽ trở thành tương lai của ngành công nghiệp phát triển game trên thế giới.

* Các thiết bị đeo tay, gắn linh hoạt trên cơ thể

Xu hướng này thường được kết hợp với Virtual Reality đã được nhắc đến ở trên, các thiết bị đeo tay hổ trợ gọi điện hay kiểm tra chỉ số sức khỏe cũng có thể trở thành những công cụ chơi game mới đầy tiềm năng.

Chúng ta có lẽ đang khá quen thuộc với việc điều khiển game thông qua keyboard, mouse trên PC hoặc touch trên các thiết bị smart phone, nhưng dù chúng ta chơi game nhiều thế nào thì cảm nhận vẫn không đủ thực như khi sử dụng các thiết bị cầm tay, hoặc gắn trên cơ thể. Một yếu tố nữa giúp chúng ta khẳng định tiềm năng của xu hướng này là game thủ luôn muốn tìm đến trải nghiệm mới lạ ở từng game khác nhau, từ chiến thuật, lối chơi, thử thách hay sự tương tác… Đó là lý do các hãng công nghệ lớn đang tìm cách kết hợp giữa các thiết bị đeo với Virtual Reality để tăng thêm sự tương tác giữa game và người chơi.

* Chơi game kết hợp vận động

Như mọi người đã biết, việc ngồi chơi game trong một khoảng thời gian dài sẽ ảnh hưởng rất nhiều đến sức khỏe con người, có thể liệt kê một số tác hại khi ngồi chơi game quá lâu như sau: đau lưng, rối loạn tư duy não bộ, hội chứng Nintendoitis, hay nguy hiểm hơn có thể kể đến suy tim, đột quỵ. Tuy nhiên, với những game phát triển theo xu hướng truyền thống, game thủ không thể tránh khỏi việc ngồi hàng giờ trước màn hình PC.

Đi ngược lại với xu hướng, Pokemon Go đã thành công vang dội khi đưa game thủ ra khỏi môi trường cô độc quen thuộc, buộc họ phải ra ngoài để tương tác với thế giới, làm quen với môi trường sống một cách gần gũi hơn. Thể loại game này đã nhận được cái nhìn thiện cảm hơn từ phía cộng đồng và xóa bỏ đi giới hạn về tuổi tác, bởi ai cũng có thể chơi game này với chỉ một chiếc điện thoại. Nó cũng mang lại ý nghĩa về mặt tâm lý, khuyến khích người dùng đến với các hoạt động tích cực.



*Pokemon Go được ra mắt trong năm 2016 thay đổi hoàn toàn lối chơi game của cộng đồng*

Bên cạnh sự thành công của Pokemon Go, còn rất nhiều đề tài khác có thể khai thác bởi công nghệ Augmented Reality như game bắn súng, nhập vai, … Tuy nhiên, đây là thử thách rất lớn mà các nhà phát triển game phải đối mặt khi vẫn còn nhiều giới hạn trong phương pháp tiếp cận và nền tảng cung cấp.

### Một số xu hướng nên thay đổi

Bên cạnh những xu hướng phát triển được cộng đồng game đón nhận tích cực trong năm qua, vẫn có những xu hướng mà game thủ không bao giờ muốn chúng tiếp tục tồn tại.

* Trailer phản ánh sai lệch ý tưởng

Sản xuất video game cũng tương tự như sản xuất phim, cũng cần có những đoạn video trailer nhằm tăng thêm sự thu hút người chơi. Tuy nhiên, có nhiều trailer điện ảnh không phản ánh đúng những yếu tố mà video game sở hữu.



*Series game Assassin's Creed - một tựa game nhập vai nổi tiếng được phát hành bởi Ubisoft*

Có thể lấy game Assassin's Creed là một minh chứng, mặc dù vẫn được đánh giá là một series video game phiêu lưu, hành động hấp dẫn nhưng bên cạnh đó, nhà phát hành cũng nhận được khá nhiều phê bình từ phía cộng đồng game thủ khi những pha hành động, chiến đấu trong game không đủ kịch tính như những gì được thể hiện qua video trailer, điều này khiến khá nhiều game thủ thất vọng.

Game trailer là công cụ để thu hút người chơi và giúp tạo nên hình tượng của game trong tư duy của game thủ. Tuy nhiên, việc lạm dụng quá nhiều vào video trailer có thể gây tác động tiêu cực đến tâm lý của người chơi.

* Quảng cáo

Đây có lẽ là điều mà người chơi ghét nhất khi chơi game. Quảng cáo làm ảnh hưởng đến trải nghiệm và mạch cảm xúc khi chơi của game thủ. Với tần số xuất hiện quá nhiều trong một khoảng thời gian ngắn thì sớm hay muộn, người chơi cũng từ bỏ cho dù game play có hấp dẫn đến đâu đi nữa.



*Fullscreen game advertisement*

Mặc dù vẫn biết đây là hình thức kiếm lời từ phía nhà phát hành game và công ty quảng cáo sản phẩm, hạn chế phần quảng cáo trong game (đặc biệt là game mobile) sẽ là một ý kiến tuyệt vời cho xu hướng game hiện nay.

Các nhà tiếp thị giờ đây sẵn sàng trả tiền để có được tấm banner thương hiệu của mình xuất hiện trên các game có số lượng người chơi đông đảo. Vị trí đặt quảng cáo càng nổi bật càng tốt, ví dụ sử dụng tên thật của xe làm các lựa chọn trong các trò game đua xe, hoặc các máy bán đồ uống trong game cũng được sử dụng logo của các nhà sản xuất nước giải khát nổi tiếng.

Dưới đây là một vài ví dụ về việc đặt quảng cáo mà không gây quá nhiều phiền toái, không ảnh hưởng đến mạch cảm xúc của người chơi mà vẫn đạt được hiệu quả nhất định:







Đặt các banner quảng cáo ngay bên trong môi trường đồ họa của game là một cách khá hữu hiệu và nên được sử dụng thay thế cho phương pháp đặt pop-up quảng cáo khiến mạch chơi game bị ngắt quãng.

* Sản phẩm độc quyền

Xu hướng này đi ngược lại với xu hướng game đa nền tảng đã được nhắc đến ở trên khi các sản phẩm game chỉ có thể chạy độc quyền trên một thiết bị chơi game nào đó. Câu hỏi cần đặt ra là tại sao phải làm vậy khi game thủ dù chơi game trên bất cứ thiết bị nào thì họ vẫn là game thủ? Và họ chỉ muốn được trải nghiệm tất cả các dòng game hay trên thị trường?

Hiện nay, xu hướng này đang dần được xóa bỏ khi có rất nhiều nhà sản xuất game chịu đầu tư vào việc chuyển đổi các tựa game sang nhiều hệ điều hành khác nhau. Việc chuyển đổi hệ điều hành cho các game nổi tiếng cũng đã tạo ra nhiều công việc hơn cho những lập trình viên đi theo hướng phát triển game.

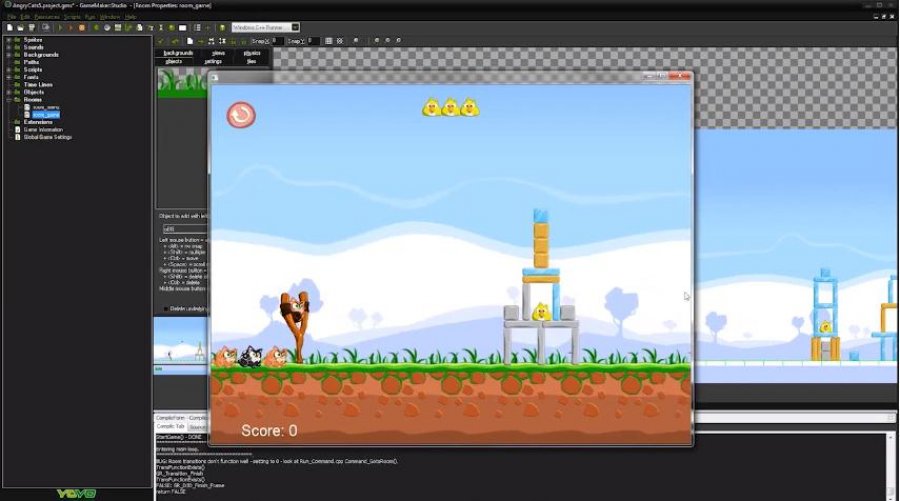


## Các công nghệ phát triển game đang được sử dụng phổ biến

Hiện nay, do sự phát triển chóng mặt của mặt hàng công nghệ smart phone dẫn đến sự phát triển nhanh chóng của thể loại mobile games. Cùng với sự phát triển đó, có vô số các lập trình viên cũng bắt đầu tham gia vào lập trình game mobile, và với những công cụ mã nguồn mở (cũng có những công cụ mã nguồn đóng) được cung cấp sẵn thì hầu như mọi người đều có thể tự thiết kế được game cho bản thân mình. Dưới đây, em xin giới thiệu một số công nghệ thiết kế game từ mức độ đơn giản đến phức tạp:

* GameMaker | YoYo Games

Với một người mới bắt đầu làm game và chưa có bất kỳ kinh nghiệm nào với lập trình, chắc chắn bạn sẽ muốn có một công cụ dễ sử dụng, GameMaker sẽ khá phù hợp và dễ sử dụng.



*Game mô phỏng Angry Bird được thiết kế với GameMaker*

GameMaker là một công cụ miễn phí dùng để tạo game mà không cần phải lập trình nhiều. Nó là một giao diện hoàn toàn kéo thả và chúng ta có thể xuất bản game của mình tới các hệ điều hành Windows, Linux, iOS, Android, … GameMaker nhắm đến việc tạo ra các Sprite-based rất dễ dàng. GameMaker còn cho phép liên kết các SDK bên ngoài, can thiệp trực tiếp vào trong code và hơn thế nữa. Tuy nhiên, chúng ta rất khó có thể thiết kế một game quá phức tạp với GameMaker, do đó nếu chúng ta muốn làm một cái gì đó kiểu như một game chiến thuật, game RPG… thì chắc hẳn chúng ta cần tìm một công cụ khác.

* Cocos2d-x

Cocos2d-x là một game engine mã nguồn mở dùng cho việc tạo ra các game 2D. Game sản xuất bằng Cocos2d-x có thể đưa lên Windows, Mac, Android, iOS, Windows phone hoặc Web. Phần lớn ngôn ngữ được sử dụng trong Cocos2d-x là C++ (và cũng có hỗ trợ các ngôn ngữ Lua script và Javascript), vì vậy game engine này có yêu cầu cao hơn, đòi hỏi người sử dụng phải nắm được các ngôn ngữ lập trình được hỗ trợ trước khi bắt tay vào thiết kế sản phẩm. Tuy nhiên, khi đã thành thạo những ngôn ngữ này, Cocos2d-x sẽ là một công cụ mạnh mẽ đáng để chúng ta khai thác.



*Game Nông trại PRO 2.0 được phát triển bằng Cocos2d-x*

Cocos2d-x được phát triển với một cộng đồng hỗ trợ rất lớn, nguồn tài liệu cung cấp cho công cụ này khá đầy đủ và nó khá phù hợp cho những lập trình viên game đang ở mức độ trung bình. Sản phẩm của đồ án này được xây dựng bằng game engine này nên chúng ta sẽ đi sâu tìm hiểu về Cocos2d-x trong những phần tiếp theo.

* Unity

Để tạo ra những game phức tạp hơn như các game 3D thì Unity có thể là một lựa chọn tuyệt vời.

Với Unity, chúng ta có thể tạo ra các game 3D và cả 2D trên hầu hết mọi nền tảng hiện có. Chúng ta cũng có thể đưa các tài nguyên được thiết kế từ các công cụ như 3Ds Max, Maya, Blender, … Unity sử dụng ngôn ngữ C# hoặc Javascript làm ngôn ngữ để viết script., những ngôn ngữ này không quá khó để sử dụng thành thạo.

Mặc dù Unity là một game engine lớn và mạnh mẽ, việc học cách sử dụng Unity lại phần nào đơn giản hơn các game engine khác nhờ rất nhiều thư viện được tích hợp sẵn, và những plugin có thể dễ dàng tích hợp vào engine, cùng với khả năng tái sử dụng những khuôn mẫu đã được thiết kế trước đó giúp các nhà phát triển tiết kiệm thời gian và chi phí hơn.



*Game xe tank được thiết kế với Unity*

Với phương pháp tiếp cận dễ dàng, Unity giúp cho các nhà phát triển game cá nhân giảm bớt rất nhiều gánh nặng về lập trình, để họ có thể tập trung hơn vào phần thiết kế game play, đồ họa, các tutorial cũng đã được cung cấp sẵn…

Với khả năng đáp ứng hầu hết mọi yêu cầu mà một nhà phát triển game có thể đặt ra, Unity cũng tồn tại những điểm yếu của riêng nó như một số phương thức thiết kế sẵn không được tối ưu, dung lượng của game sau khi build lớn hơn rất nhiều so với việc sử dụng những game engine khác, phiên bản miễn phí của Unity không đầy đủ những tính năng mong muốn trong khi bản có phí lại có giá thành quá cao, … Do đó, trước khi lựa chọn sử dụng Unity làm công cụ phát triển game, chúng ta cũng cần có những cân nhắc trước khi quyết định công cụ nào phù hợp.

* Unreal Engine

Unreal Engine được phát triển bởi Epic Games, được cho ra mắt vào năm 1998 với thể loại game bắn súng góc nhìn thứ nhất. Mặc dù mục đích ban đầu của Unreal Engine là hướng đến thể loại game bắn súng góc nhìn thứ nhất, nó cũng đã thành công trong một số thể loại game khác như MMORPGs, RPG, …

Unreal Engine sử dụng hoàn toàn ngôn ngữ C++, nên một đặc trưng dễ thấy của game engine này là hiệu suất rất cao và khả năng tương thích với nhiều nền tảng khác nhau, đó là lý do mà công cụ này được sử dụng bởi rất nhiều nhà phát triển game hiện nay. Tuy nhiên, nó đòi hỏi rất nhiều kỹ năng từ những nhà phát triển game muốn sử dụng.



*Unreal Engine được sử dụng để tạo ra những game có yêu cầu chất lượng đồ họa cao*

Ngày nay, Unreal Engine 4 đã hỗ trợ rất nhiều tính năng mạnh mẽ:

* Hỗ trợ DirectX 11 & 12 rendering.
* Quản lý mã nguồn với Github được tích hợp sẵn.
* Có thể thiết kế game bằng Blueprint để thay thế code.
* Hỗ trợ AI.
* Giả lập sẵn môi trường vật lý.
* Chất lượng đồ họa cao.
* Hỗ trợ thực tế ảo.

Ngoài ra còn rất nhiều tính năng khác của Unreal Engine mà chúng ta không thể liệt kê hết ở đây.

Ngày nay, Unreal Engine không những được sử dụng trong việc thiết kế game mà còn được áp dụng để thiết kế các mô hình kiến trúc với khả năng đồ họa cao mà nó mang lại. Unreal Engine có thể làm được nhiều thứ hơn chúng ta nghĩ, nhưng nó đặt ra khá nhiều yêu cầu về mặt trình độ và kỹ năng cho người sử dụng.

## Tổng quan về game engine

Game engine là một framework dùng cho công việc phát triển game, nó giúp đỡ những nhà phát triển game với những phần khung mà tất cả mọi game đều có: graphics, audio, game logic. Trên thực tế, đây là những thành phần mà tất cả phần mềm đều có khi mà chúng ta xây dựng những game AAA, game đơn giản như xếp gạch, hay ứng dụng Calculator. Tuy nhiên, những game engine hiện đại sỡ hữu nhiều tính năng chuyên dụng hơn, bao gồm các framework xử lý Physics, AI, networking, tối ưu hóa, và hơn thế nữa.

Cùng nhìn về lịch sử của ngành công nghiệp phát triển game, hầu như mọi game đều được thiết kế và xây dựng nên từ con số không.



*Game Super Mario Brothers được phát triển từ những năm 80 của thế kỷ XX*

Lấy ví dụ game Super Mario Brothers chơi trên máy cầm tay Nintendo, nó được thiết kế từ những trang được vẽ bởi Miamoto Shigeru và được tải vào game tại đúng vị trí. Chúng ta có thể thấy nền gạch, những cái ống, những đồng tiền, … rất nhiều đối tượng khác nhau trong level đầu tiên, và tất cả chúng được tải lên bộ nhớ và hiển thị chúng tại nơi chúng ta cần hiển thị. Sau khi những đối tượng này hoàn tất nhiệm vụ của nó, chúng ta cần xóa bỏ chúng ra khỏi bộ nhớ ngay lập tức. để chuẩn bị tải hình ảnh cho khu vực tiếp theo. Vì vậy chúng ta cần đảm bảo rằng bất cứ khi nào chúng ta tạo ra một đối tượng mới, chúng ta phải tìm cách xóa những đối tượng phía sau từng cái một.

Điều này sẽ dễ dàng hơn nhiều nếu chúng ta có một hệ thống có thể quản lý những đối tượng trong game, và tự động xóa chúng ra khỏi bộ nhớ ngay khi game cần chuẩn bị tải level tiếp theo. Để hệ thống này chạy trong suốt quá trình game được thực thi sẽ tiêu hao một lượng bộ nhớ đáng kể, nhưng nó có thể tiết kiệm khá nhiều thời gian cho công việc phát triển game, đó là một sự trao đổi có lợi cho những nhà phát triển. Tất cả mọi game đều có những đối tượng, nhân vật, và kẻ địch, … và nó sẽ rất tuyệt vời nếu hệ thống quản lý tài nguyên này được tạo ra để sử dụng cho tất cả mọi game chứ không phải một game duy nhất. Khi đó, những game như Mario, Mega Man hay Contra sẽ không cần thiết phải viết lại những chức năng giống nhau. Đó cũng là mục đính chính của việc tạo ra một game engine, nó cung cấp những công cụ, nền tảng mà đa số các game yêu cầu để xử lý. Sử dụng game engine đồng nghĩa với việc những nhà phát triển game không cần "tái tạo lại bánh xe" mỗi lần bắt đầu một dự án game mới.

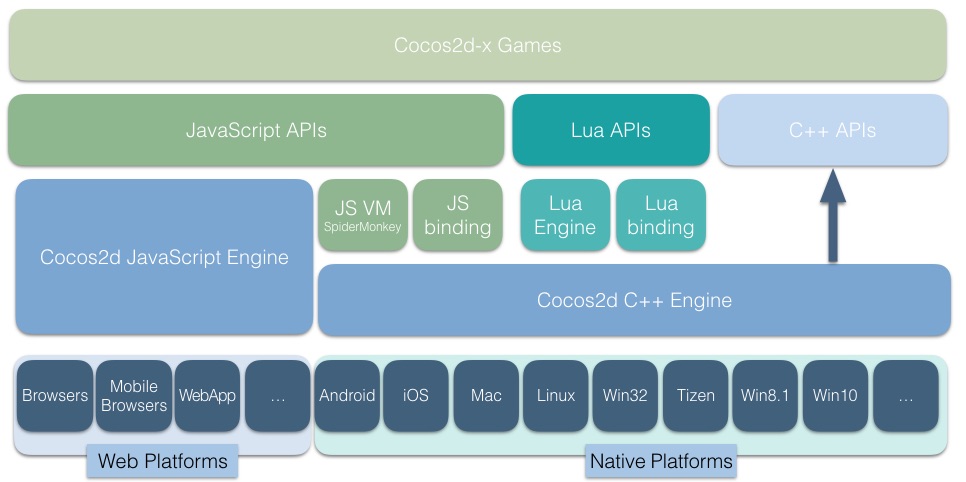
Những game engine được phát triển bởi bên thứ ba (Third party game engine) chỉ trở nên phổ biến rộng rãi khi nhu cầu chơi game 3D bắt đầu được ưa chuộng và những công nghệ đang phát triển trở nên phức tạp.

Ngày nay, nhiều game studio đã bắt đầu sử dụng game engine được phát triển bởi chính họ để tăng tốc quy trình phát triển game.

# TÌM HIỂU VỀ COCOS2D-X GAME ENGINE

## Cocos2d-x là gì?

Được phát triển từ năm 2010, Cocos2d-x là một game engine mã nguồn mở và đa nền tảng với những tính năng mạnh mẽ, cho phép những nhà phát triển tạo ra những game 2D, những ứng dụng đồ họa một cách dễ dàng.



*Kiến trúc của Cocos2d-x framework*

**Lý do chọn sử dụng Cocos2d-x cho đồ án:**

* Cocos2d-x được phát triển với C++ API hiện đại (chuẩn C++11).
* Đa nền tảng - desktop và mobile.
* Hỗ trợ test và debug game trên desktop và xuất bản chúng trên mobile hoặc desktop.
* Cung cấp một số lượng lớn các API với các tính năng bao gồm sprites, actions, particles, timers, events, sound, file I/O, 3D.
* Nhanh, miễn phí, dễ sử dụng, cộng đồng hỗ trợ tốt, nhiều nguồn tài liệu.

## Những thành phần cơ bản của Cocos2d-x

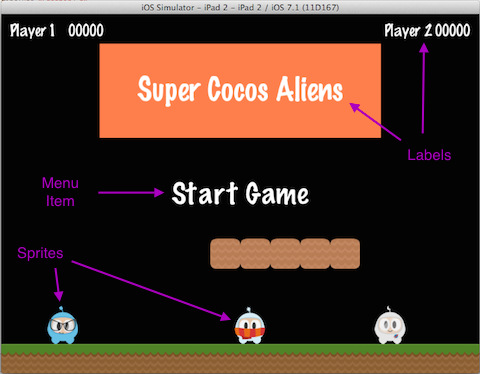
Như chúng ta đã biết, Game engine được tạo ra nhằm mục đích tiết kiệm thời gian phát triển game, và nó thường tối ưu hơn các thuật toán do các nhà phát triển cá nhân tự định nghĩa. Và Cocos2d-x game engine cung cấp cho chúng ta những API đơn giản cho việc thiết kế game đa nền tảng, những nhà phát triển game không cần quá bận tâm vào việc tìm hiểu phần kỹ thuật bên trong.

Cocos2d-x cung cấp các loại đối tượng như Scene, Transition, Sprite, Menu, Audio, … và rất nhiều thành phần khác cần thiết cho mục đích thiết kế game. Và những thành phần thường xuyên được sử dụng nhất trong game, phần lõi không thể thiếu chính là Scene, Node, Sprite, Menu và Action. Thử nhìn lại một số game mà chúng ta từng chơi, chúng ta sẽ thấy tất cả các thành phần chính này đều xuất hiện trong đó.

Hãy cùng nhìn qua một ví dụ đơn giản:



Trên đây là một hình ảnh của một frame được render trong một demo game. Có khá nhiều thành phần trong đó, hãy cùng xác định những thành phần đã được sử dụng để xây dựng nó:



Đầu tiên chúng ta có thể thấy một menu, một vài sprite và label, hiện tại nhìn chúng có vẽ khá tương tự nhau. Tuy nhiên, mỗi thành phần đảm nhận một chức năng khác nhau mà chúng ta sẽ tìm hiểu chúng trong phần sau.

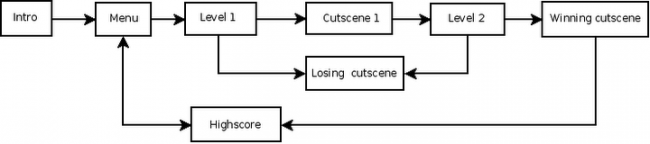
* Director

Cocos2d-x sử dụng khái niệm Director - người đạo diễn, giống như một bộ phim. Một game được phát triển sử dụng Cocos2d-x chỉ có một đối tượng Director duy nhất, nó điều khiển hành động của các đối tượng trong game, theo một thứ tự nhất định mà nhà phát triển game đặt ra.

Từ khái niệm Director, có thể suy ra những nhà phát triển game sử dụng Cocos2d-x sẽ được xem như những Executive Producer và họ nói cho Director biết những gì cần làm.

Một nhiệm vụ thường thấy của Director là điều khiển các Scene, chuyển đổi và thay thế chúng. Director trong Cocos2d-x được định nghĩa theo Singleton design pattern (chỉ tồn tại duy nhất một đối tượng Director tại một thời điểm) và nó có thể được truy cập từ bất kỳ vị trí nào trong code.

Dưới đây là một ví dụ cơ bản về dòng chảy của game. Đối tượng Director đảm nhận nhiệm vụ chuyển đổi qua lại giữa các Scene theo quyết định của nhà phát triển:



Những nhà phát triển quản lý đối tượng Director, và họ sẽ quyết định những gì xảy ra, khi nào và như thế nào.

* Scene

Trong một game, chúng ta có thể thường thấy một Main Menu, một vài game level, và một cảnh kết thúc. Để tổ chức chúng thành các phần riêng biệt, chúng ta sẽ sử dụng Scene, giống như những cảnh trong phim, chúng tách biệt với những phần khác của câu chuyện. Nếu chúng ta áp dụng tư tưởng này vào game, chúng ta sẽ thiết kế một vài Scene trong game với độ phức tạp của mỗi Scene tùy vào quyết định của nhà phát triển.

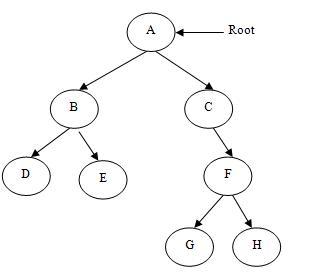
Dưới đây là một ví dụ mà chúng ta đã thấy trong phần trước:



Đây là một Main Menu mà nó là một Scene độc lập. Trong Scene này, chúng ta có một vài đối tượng ăn khớp với nhau. Scene được vẽ bởi Renderer, renderer chịu trách nhiệm vẽ các Sprite, các đối tượng lên màn hình dựa trên khái niệm Scene graph.

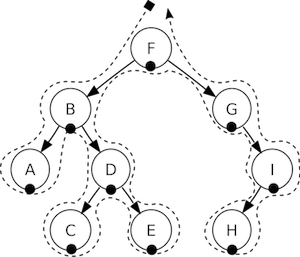
* Scene Graph

Scene graph là một cấu trúc dữ liệu dùng để sắp xếp đồ họa trong Scene. Một Scene graph chứa những Node trong cây (scene graph tổ chức theo cấu trúc dữ liệu cây).



Để hiểu được cách mà một Scene được vẽ lên màn hình, chúng ta cần hiểu được cách hoạt động của Scene graph.

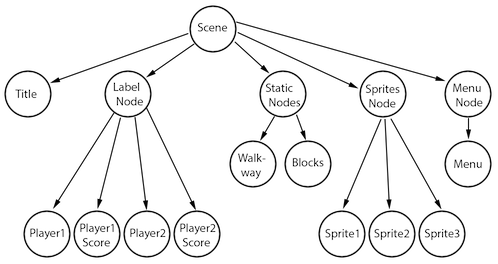
Khi chúng ta thêm những Node, những Sprite hoặc Animation vào Scene, chúng ta mong muốn chúng được vẽ lên màn hình, nhưng đôi lúc, một số trong chúng lại bị ẩn phía sau background và chúng ta muốn chúng được đặt ra phía trước background. Điều này không thành vấn đề, chúng ta chỉ cần tổ chức các đối tượng trong Scene, thêm chúng vào Scene graph tại đúng vị trí, đúng thứ tự duyệt cây. Và thứ tự duyệt cây của Scene graph trong Cocos2d-x bắt đầu từ phía bên trái, sau đó là đến root node, cuối cùng là sang bên phải của cây (LNR).



Sau khi Scene graph được duyệt xong, renderer sẽ vẽ chúng lên màn hình.

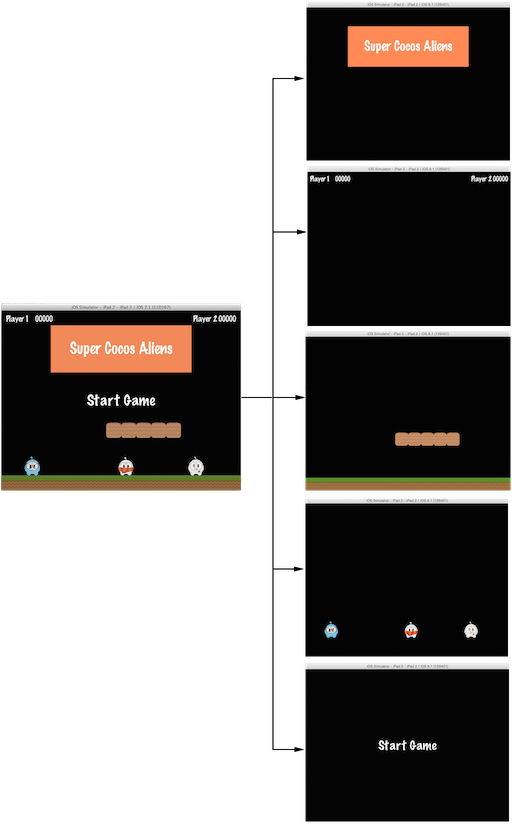


Cùng xem một ví dụ về main menu được vẽ lên màn hình theo cơ chế hoạt động của Scene graph:



Để dễ hình dung hơn, chúng ta sẽ sử dụng khái niệm **z-order.** Những node nằm phía bên trái sẽ có **z-order** bé hơn các node phía bên phải. Như vậy, các node có **z-order** càng lớn sẽ được vẽ sau cùng.

Dựa theo khái niệm này, chúng ta xem một Scene giống như một tập hợp các Node, bây giờ chúng ta sẽ đặt **z-order** cho các node trong Scene, và đây là một ví dụ về thứ tự render của các node theo **z-order**:

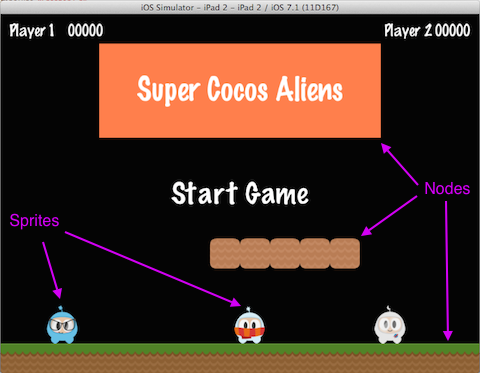


Đối tượng Scene được tạo ra như một môi trường để gắn các đối tượng Node lên đó, và mỗi đối tượng Node sẽ được chỉ định một **z-order** khác nhau khi được đưa vào Scene graph. Để tổ chức Scene graph, trong Cocos2d-x chúng ta sử dụng phương thức ***addChild()***:



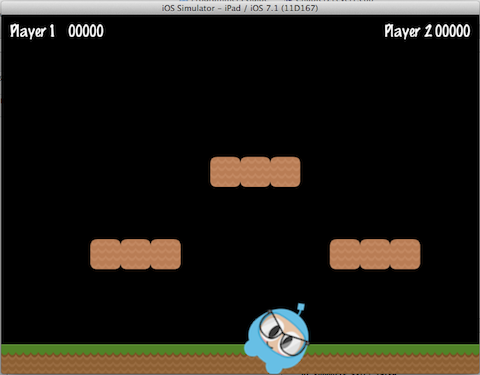
* Sprites

Tất cả game 2D đều có những đối tượng Sprite, đôi khi chúng ta không nhận ra chúng trong game nhưng chúng tồn tại. Những đối tượng Sprite có thể di chuyển xung quanh bên trong màn hình, chúng ta có thể điều khiển chúng. Thông thường, nhân vật chính trong game là một đối tượng Sprite. Nó chỉ là một đối tượng Sprite khi chúng chuyển động bên trong Scene, nếu không thì nó chỉ là một Node. Nhìn vào ví dụ bên dưới, chúng ta có thể phân biệt được những đối tượng nào là Sprite, những đối tượng nào chỉ là Node:



Một đối tượng Sprite có thể dễ dàng được tạo ra bằng Cocos2d-x API, và chúng có những thuộc tính có thể chỉnh sửa được như: position, rotation, scale, color, …

Kết quả của đoạn code này là một Sprite được tạo ra trên Scene với vị trí tính theo pixel trên trục tọa độ là (500, 0); nó lệch một góc 40 độ và có độ lớn gấp 2 lần trong file mysprite.png; ngoài ra, điểm neo của nó là (0, 0); như hình bên dưới:



* Actions

Tạo ra một Scene với những đối tượng bên trong nó chỉ mới là bước khởi tạo. Để một game trở thành một game thực sự, chúng ta cần làm cho những đối tượng trong Scene di chuyển. Action là một phần không thể thiếu của tất cả các game. Actions cho phép các Node thực hiện các phép biến đổi theo thời gian và không gian. Tạo ra một Action trong Cocos2d-x khá đơn giản:



* Sequences and Spawns

Với việc cho các đối tượng Sprite di chuyển trên màn hình, như vậy đã đủ cho chúng ta tạo ra một game? Sẽ thế nào nếu một đối tượng Sprite thực hiện cùng lúc nhiều Action? Như thế, chúng ta cần một cách khác để xử lý.

Giống như tên gọi của nó, Sequence là một đối tượng quản lý nhiều đối tượng Action bên trong và sắp xếp chúng theo một thứ tự nhất định, và chúng ta hoàn toàn có thể đảo ngược trật tự sắp xếp của chúng.

Cùng xem một ví dụ về luồng của Sequence được áp dụng cho 1 Sprite để di chuyển nó dần dần:

http://www.cocos2d-x.org/docs/programmers-guide/basic_concepts-img/2_sequence_scaled.png

Chuỗi Action này có thể dễ dàng được tạo ra trong Cocos2d-x code:

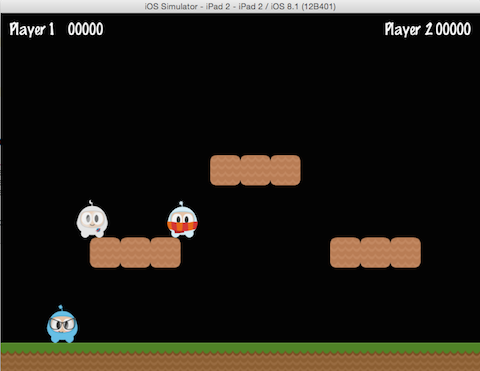
Ví dụ này thực hiện chạy một Sequence, theo một thứ tự mà nhà phát triển game sắp đặt sẵn, nhưng chúng ta sẽ làm như thế nào để chạy nhiều Action cùng một thời điểm? Chúng ta có thể đã từng nghe đến khái niệm multi-threading, một khái niệm phức tạp và khó để thực hiện trong một project lớn, nhưng Cocos2d-x đã cung cấp sẵn cho chúng ta kiểu đối tượng Spawn, đối tượng này sẽ nhận vào tất cả những Action mà nhà phát triển yêu cầu chúng chạy trong cùng một thời điểm, nhưng không đảm bảo rằng chúng sẽ kết thúc cùng một thời điểm.



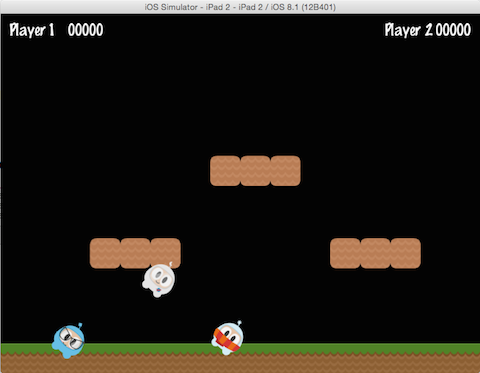
Vậy trường hợp nào chúng ta sẽ sử dụng Spawn? Giả sử nhân vật chính của chúng ta thực hiện nhiều hành động khác nhau khi nhập được power-up, tăng kích thước và phát ra một chấn động nhỏ. Cũng có thể khi kết thúc một level, chúng ta có nhiều việc cần thực hiện cùng lúc như cập nhật số tiền kiếm được trong game và quy đổi điểm số hiện tại thành năng lượng…

* Mối quan hệ Parent và Child

Cocos2d-x sử dụng mối quan hệ Parent và Child, điều này có nghĩa những thay đổi về thuộc tính của Parent Node sẽ tác động đến Children Node của nó. Thử nghiệm với một Sprite và một Sprite có Children Node:



Với những đối tượng có vai trò là Children, thay đổi thuộc tính rotation của đối tượng Parent Node sẽ làm thay đổi thuộc tính rotation của tất cả Children:



Tuy nhiên, không phải tất cả những sự thay đổi thuộc tính của Parent Node đều tác động xuống Children Node. Ví dụ thay đổi tọa độ điểm neo (anchor point) chỉ tác động chỉ ảnh hưởng đến các phép biến đổi (scale, position, rotate, skew, …) mà không tác động đến thuộc tính position của Children Node. Trên thực tế, Children Node khi được thêm vào đối tượng Parent Node thì luôn được đặt tại tọa độ (0, 0) tính từ góc trái dưới của đối tượng Parent Node.

* Tổng kết phần khái niệm Cocos2d-x cơ bản

Chúng ta vừa đi qua những khái niệm tổng quan của những thành phần quan trọng trong Cocos2d-x framework. Trong những phần tiếp theo, chúng ta sẽ đi vào tìm hiểu rõ hơn từng thành phần mà Cocos2d-x cung cấp.

## Sprites

### Sprite là gì?

Một Sprite là một hình ảnh 2D có thể hoạt động hoặc bị biển đổi bởi sự thay đổi các thuộc tính, bao gồm **rotation**, **position**, **scale**, **color**, …

### Tạo đối tượng Sprite với Cocos2d-x

Có nhiều cách để tạo ra một đối tượng Sprite phụ thuộc vào những gì chúng ta muốn đạt được. Chúng ta có thể tạo một Sprite từ một file ảnh với nhiều định dạng khác nhau: **PNG**, **JPEG**, **TIFF**, …

Cách đơn giản nhất để tạo một Sprite là sử dụng phương thức ***create()*** và truyền vào tên file (đường dẫn tương đối trong thư mục Resources của project Cocos2d-x).

Nếu trong thư mục Resources của project Cocos2d-x có file ***mysprite.png,*** một đối tượng Sprite sẽ được tạo ra.



Phương thức này sẽ tạo ra một Sprite mới với kích thước hình ảnh 2D đúng bằng kích thước hiển thị của file ảnh được chọn.

### Tạo đối tượng Sprite từ Sprite Sheet

Một Sprite Sheet là một cách để kết hợp nhiều Sprite lại vào trong một file. Sử dụng Sprite Sheet sẽ giúp tối ưu hóa dung lượng bộ nhớ cần thiết để lưu trữ tất cả Sprite bên trong nó.



Khi sử dụng một Sprite Sheet, chúng ta cần tải chúng vào trong một đối tượng SpriteFrameCache. SpriteFrameCache là một cache class, nó lưu trữ những đối tượng SpriteFrame đã được thêm vào nó, để tăng tốc truy cập trong tương lai. Một SpriteFrame được tạo ra một lần và được lưu trữ bên trong SpriteFrameCache.

Chúng ta nên khởi tạo một đối tượng SpriteFrameCache càng sớm càng tốt, có thể là trong file **AppDelegate.cpp**:

Bây giờ, chúng ta đã có một Sprite Sheet được lưu trữ bên trong đối tượng SpriteFrameCache, từ đó chúng ta có thể tạo ra những Sprite bằng cách sử dụng cache này.

Với cách này, chúng ta thu được kết quả tương tự nhưng tiết kiệm được dung lượng bộ nhớ sử dụng để lưu trữ nhiều Sprite khác nhau.

### Công cụ tạo ra các Sprite Sheet

Tạo ra những Sprite Sheet một cách thủ công rất khó khăn. May mắn thay, có một số công cụ hỗ trợ chúng ta làm điều này một cách tự động.

• Cocos Studio

• ShoeBox

• Texture Packer

• Zwoptex

### Thao tác với Sprite

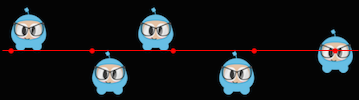
Sau khi tạo ra một đối tượng Sprite, chúng ta có thể truy cập để thao tác với các thuộc tính của nó.

* Anchor Point và Position

Anchor Point (điểm neo) là một tọa độ dùng để xác định điểm nào trên Sprite sẽ được sử dụng để thiết lập vị trí của nó. Thuộc tính Anchor point bị thay đổi chỉ tác động đến những thuộc tính có thể sử dụng phép biến đổi (bao gồm **scale**, **rotation**, **skew**). Anchor point sử dụng hệ trục tọa độ đặt ở góc trái dưới của một Sprite, điều này có nghĩa khi xác định giá trị X và Y của anchor point, chúng ta tính từ góc trái dưới của Sprite. Một Node trong Cocos2d-x được tạo ra sẽ có Anchor point mặc định là (**0.5**, **0.5**).

Để thay đổi Anchor Point của một Node, chúng ta đơn giản sử dụng phương thức ***setAnchorPoint():***

Hình bên dưới mô tả sự thay đổi của một Sprite sau 5 lần thay đổi Anchor point:



* Rotation

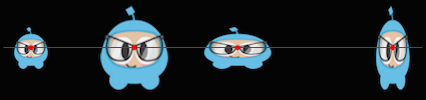
Thay đổi thuộc tính **rotation** của Sprite sẽ làm thay đổi góc quay của đối tượng được vẽ trên màn hình.

Giá trị dương truyền vào sẽ quay Sprite theo chiều kim đồng hồ, ngược lại, giá trị âm sẽ quay Sprite theo ngược chiều kim đồng hồ. Giá trị mặc định của thuộc tính rotation khi một Sprite được tạo ra là **0**.



* Scale

Thuộc tính **scale** đại diện cho tỉ lệ kích thước của Sprite so với kích thước gốc. Giá trị mặc định là **1.0** cho cả X và Y.



* Color

Thuộc tính **color** của một Sprite có kiểu dữ liệu **cocos2d::Color3D.** Color3B chứa 3 thành phần **red**, **green**, và **blue** mà mỗi thành phần sử dụng 1 byte bộ nhớ (0 - 255). Cocos2d-x còn định nghĩa sẵn một số màu cơ bản để chúng ta dễ lựa chọn hơn, ví dụ Color3B::White hay Color3B::Red.



* Opacity

Thuộc tính **opacity** đặc trưng cho sự trong suốt của một đối tượng trên màn hình. Thuộc tính này có phạm vi giá trị mong đợi là 0 đến 255, khi mà 0 nghĩa là đối tượng hoàn toàn trong suốt, tương tự như việc bị ẩn đi khỏi màn hình.



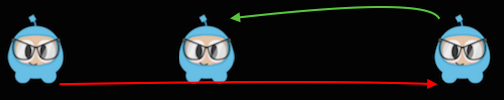
## Actions

Những đối tượng Action đúng như tên gọi của chúng, nó làm cho các đối tượng Node trên Scene hoạt động, nhưng thực chất, chúng làm thay đổi các thuộc tính của Node mà nó được gắn vào. Những đối tượng Action cho phép thực hiện các phép biến đổi trên các thuộc tính của một Node, và tất cả các class thừa kế từ class Node đều có thể thực hiện một hoặc nhiều Action. Ví dụ:



### Phân biệt Action By và Action To

Với những Action cơ bản, chúng được chia thành 2 loại: By và To; dựa trên sự khác biệt của kết quả mà chúng đem lại. Một By-type Action mang tính tương đối so với trạng thái hiện tại của Node. Một To-type Action là tuyệt đối, nghĩa là chúng không quan tâm đến trạng thái hiện tại của Node. Chúng ta cùng xem một ví dụ:



Đầu tiên, một Sprite được tạo ra tại tọa độ (**200**, **256**) theo hệ trục tọa độ của Cocos2d-x quy định. Một Sequence được tạo ra với một chuỗi các Action cơ bản: di chuyển qua phải **500** pixel, trì hoãn 1 giây, di chuyển đến đúng vị trí (**300**, **256**).

### Chạy các Action đơn giản

* Move

Sử dụng action Move để di chuyển một Node trong một khoảng thời gian:

* Rotate

Sử dụng action Rotate để quay một Node theo chiều kim đồng hồ trong một khoảng thời gian:



* Scale

Sử dụng action Scale để thay đổi tỉ lệ kích thước của một Node trong một khoảng thời gian:



* Fade In / Out

Sử dụng action Fade để thay đổi độ mờ nhạt của một đối tượng Node:



* Animate

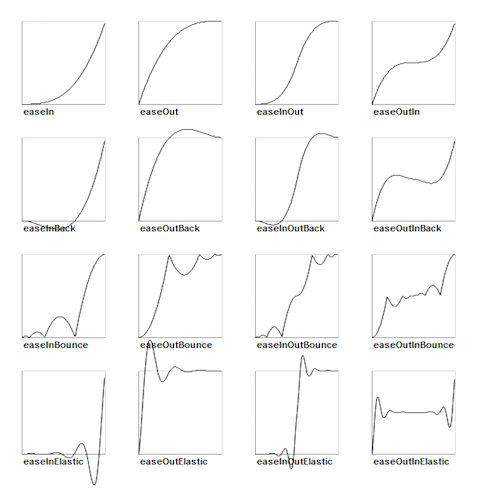
Sử dụng action Animate để làm cho đối tượng Sprite trở nên sinh động hơn. Đối tượng Animate sẽ quản lý một đối tượng khác bên trong là Animation, và đối tượng Animation sẽ lưu trữ một tập hợp các khung hình (frames) khác nhau của một hoạt động của đối tượng Sprite nào đó, những khung hình này sẽ được đối tượng Animate thay thế lần lượt trong một khoảng thời gian:



* Easing

Easing là được sử dụng để thiết lập gia tốc cho sự biến đổi các thuộc tính khi thực hiện Action hoặc Animation. Một điều cần chú ý là nó liên quan đến tốc độ, các Ease Action luôn luôn bắt đầu và kết thúc tại cùng thời điểm. Ease Action là một cách tốt để làm giả tính chất vật lý trong game, khi chúng ta có thể thêm vào một số hiệu ứng giả lập vật lý nhưng không làm tăng độ phức tạp một cách đáng kể.

Dưới đây là một số đồ thị đặc trưng cho mỗi loại Ease Action khác nhau:



Cocos2d-x hỗ trợ hầu hết các loại Ease Action có trong bảng đồ thị trên, và chúng khá dễ dàng được thực hiện trong code. Thử lấy một ví dụ về một đối tượng Sprite và làm nó tung lên:



* Sequence

Một Sequence là một tập hợp các đối tượng Action được sắp đặt theo thứ tự vào trước thực hiện trước, mọi đối tượng có kiểu thừa kế từ class Action đều có thể được thêm vào Sequence, thậm chí cả một **callback function**. Cocos2d-x có một kiểu đối tượng CallFunc sẽ xác định một hàm được trỏ đến. Điều này cho phép chúng ta thêm vào nhiều chức năng hơn cho một Sequence, ví dụ sau khi di chuyển một đoạn đường thì thực hiện tấn công.



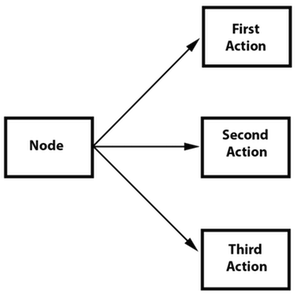
Dưới đây là một ví dụ về cách tạo và thực hiện một đối tượng Sequence:

Đối tượng Sequence này sẽ khiến đối tượng mySprite thực hiện một chuỗi hành động:

**Jump** -> **callbackJump()** -> **Rotate** -> **callbackRotate()**

* Spawn

Spawn Action cũng khá giống với Sequence Action, ngoài trừ việc nó chạy tất cả Action mà nó quản lý cùng một thời điểm.



Spawn Action cho ra kết quả hoàn toàn tương tự với việc cho Node chạy nhiều Action. Tuy nhiên, điểm mạnh mà Spawn Action mang lại là chúng ta có thể đặt một Spawn vào một Sequence để đạt được những hiệu ứng đặc biệt hơn. Kết hợp được Spawn và Sequence là một thế mạnh của Cocos2d-x.

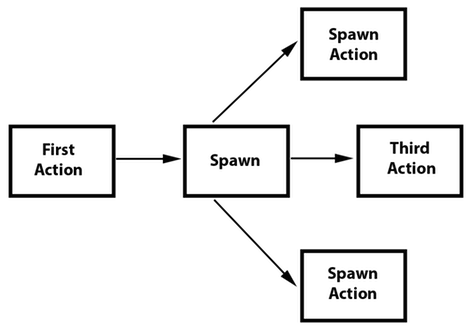
Dưới đây là một ví dụ đơn giản về cách sử dụng Spawn Action:



Và đây là cách mà chúng ta kết hợp Spawn với Sequence:



Với cách thực hiện này, chúng ta có thể biểu diễn dưới dạng sơ đồ khối như sau:



## Scenes

### Scene là gì?

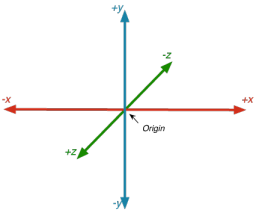
Một Scene là một container object có khả năng chứa những Node khác bên trong nó. Scene đảm nhiệm công việc thực hiện logic game và vẽ nội dung game lên màn hình từng frame một. Chúng ta cần ít nhất 1 Scene để xây dựng game lên trên nó. Với tư tưởng của Cocos2d-x, một Scene cũng giống như 1 cảnh quay nào đó trong phim, khi một Scene đang được thực thi, người dùng sẽ nhìn thấy những gì xảy ra bên trong nó. Chúng ta có thể thêm vào nhiều Scene khác nhau cho game và có thể thực hiện chuyển đổi qua lại giữa các Scene.

### Tạo ra một Scene với Cocos2d-x

Nó khá là đơn giản để tạo ra một Scene mới trong Cocos2d-x:

Chúng ta cần nhớ rằng, các đối tượng được thêm vào Scene sẽ hoạt động dựa trên khái niệm **Scene Graph** và **z-order** mà chúng ta đã tìm hiểu trong phần trước.

Bây giờ, chúng ta sẽ xây dựng một Scene đơn giản. Lưu ý rằng Cocos2d-x sử dụng hệ trục tọa độ theo nguyên tắc bàn tay phải.



Điều này có nghĩa tọa độ (**0**, **0**) sẽ ứng với góc trái dưới của màn hình. Do đó, khi tạo ra các đối tượng trong game, chúng ta cần tính toán trước vị trí bắt đầu phù hợp cho chúng.

Khi chúng ta thực thi đoạn code này, Cocos2d-x tạo ra một Scene với một Label và một Sprite, đó chỉ mới là bước khởi đầu.

### Scene Transition

Chúng ta sẽ cần chuyển Scene trong game. Có thể là do bắt đầu một level mới, hoặc do một Scene đã thực hiện xong công việc của nó. Cocos2d-x cung cấp cho chúng ta một số cách khác nhau để thực hiện chuyển đổi Scene.

Việc đầu tiên chúng ta cần làm là tạo ra một Scene mới:

Sử dụng phương thức ***runWithScene()*** là một cách để chúng ta bắt đầu game. Chúng ta chỉ nên sử dụng phương thức này cho lần tạo Scene đầu tiên.

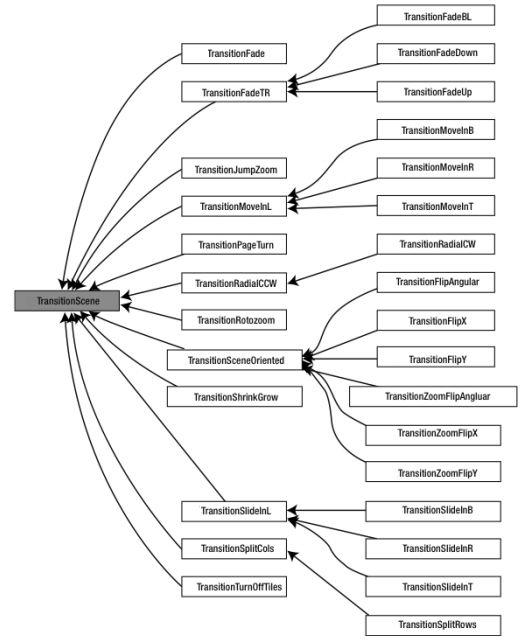
Phương thức ***replaceScene()*** sẽ thay thế Scene đang chạy:

Phương thức ***pushScene()*** sẽ khiến game hoãn việc thực thi Scene hiện tại, và đẩy vào **Scene Stack** một Scene mới:

Ngược lại với phương thức ***pushScene(),*** phương thức ***popScene()*** sẽ hủy Scene đang chạy, đưa nó ra khỏi **Scene Stack,** và thực hiện tiếp Scene bị trì hoãn bởi phương thức ***pushScene()*** đã được gọi trước đó:

Chúng ta có một cách đặc biệt hơn để chuyển đổi Scene, đó là thêm vào một số hiệu ứng trong quá trình chuyển đổi. Dưới đây là một số ví dụ:

Ngoài ra, chúng ta còn có nhiều hiệu ứng khác với các đối tượng Transition khác nhau.



## Các thành phần UI

Thử nhìn lại một số ứng dụng mà chúng ta đã sử dụng, có thể thấy rằng tất cả chúng đều có những UI widgets. Những thành phần UI này thường thấy nhiều ở các Main Menu của game, chúng không tham gia vào việc tạo nên game play, nhưng nó là một thành phần không thể thiếu của một game.

### Widgets

Cocos2d-x cung cấp cho chúng ta một bộ UI widgets đơn giản và dễ sử dụng trong các project. Tuy trông chúng khá là đơn giản, nhưng để tạo ra một một bộ UI widgets như thế cũng tốn khá nhiều thời gian cho các nhà phát triển, nên Cocos2d-x UI giúp chúng ta tiết kiệm thời gian thiết kế hơn cho việc tự xây dựng bộ UI widgets cho riêng mình.

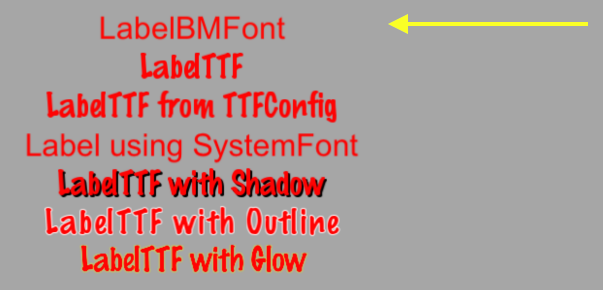
### Labels

Cocos2d-x cung cấp đối tượng Label có thể được tạo ra với các kiểu **true type, bit map** hoặc các font có sẵn trong hệ thống. Class Label có thể xử lý tất cả các dạng Label cần thiết.

* Label BMFont

BMFont là một kiểu label sử dụng bitmap font. Các ký tự trong một bitmap font được tạo ra bởi một ma trận các dấu chấm. Nó rất nhanh và dễ sử dụng, nhưng nó không có khả năng mở rộng bởi nó đòi hỏi một font khác nhau cho mỗi kích thước các ký tự. Mỗi ký tự trong một Label là một Sprite riêng biệt. Điều này có nghĩa mỗi ký tự có thể bị thay đổi các thuộc tính **rotate**, **scale**, …

Để tạo ra một BMFont Label cần có 2 files: một file **.fnt** và một file hình ảnh biểu diễn mỗi ký tự trong định dạng **.png.** Chúng ta có thể tự tạo ra những file này bằng công cụ Glyph Designer. Dưới đây là một ví dụ về việc tạo ra BMFont Label trong Cocos2d-x:

Tất cả các ký tự trong tham số kiểu **string** cần được tìm thấy trong file **.fnt**, nếu không tìm thấy, nó sẽ không được vẽ lên màn hình.

* Label TTF

**True Type Font** khác với **bitmap font** mà chúng ta đã tìm hiểu ở trên. Với **true type font,** nét vẽ phác thảo của font được vẽ. Đây là sự thuận tiện khi không cần phải có một file cho mỗi font riêng biệt, chúng ta có thể sử dụng một **true type font** cho nhiều kích thước và màu sắc khác nhau. Tạo ra một đối tượng sử dụng **true type font** khá đơn giản, chúng ta chỉ cần xác định một file **.ttf**, chuỗi văn bản và kích thước của văn bản:

Mặc dù nó được sử dụng linh hoạt hơn **bitmap font,** vẽ **true type font** chậm hơn khá nhiều, việc thay đổi các thuộc tính như **font face** hay **size** rất tốn chi phí.

Nếu chúng ta cần tạo ra nhiều đối tượng Label **true type font** có cùng thuộc tính, chúng ta có thể tạo một đối tượng TTFConfig để lưu trữ các thuộc tính chung của các TTF Label:



* Label SystemFont

**System Font** là một kiểu Label sử dụng font mặc định của hệ thống với kích thước mặc định. Điều này có nghĩa **System Font** không có những thuộc tính có thể thay đổi.



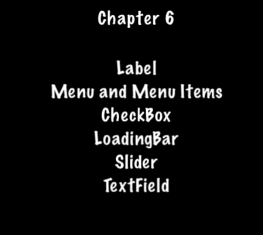
### Menu và các Menu Item

Tất cả chúng ta có thể đã quá quen thuộc với Menu, chúng ta thấy nó trên tất cả những ứng dụng chúng ta sử dụng. Trong game, chúng ta có thể sử dụng Menu để chuyển hướng đến game options. Menu thường chứa những button như button Play, Quit, Settings, About, và có thể cả những đối tượng Menu khác. Một đối tượng Menu là một kiểu đặc biệt của Node. Chúng ta có thể tạo ra một Menu rỗng như sau:

Và chúng ta có thể gắn các Menu Item lên đó, như các button Play, Quit, Settings đã nói đến ở trên. Một Menu không chứa các Menu Item thường không có ý nghĩa.

Menu Item thường có hai trạng thái: **normal** và **selected.** Khi chúng ta click hoặc touch Menu Item, một callback được kích hoạt, chúng ta có thể tưởng tượng chúng như một phản ứng dây chuyền, touch vào một Menu Item và nó chạy một đoạn code mà chúng ta đã chỉ định.

Một Menu cũng có thể được tạo từ một **vector** các Menu Item:



Cocos2d-x sử dụng chuẩn C++11, nên chúng ta có thể sử dụng **lambda function** như một **callback**.



### Buttons

Tất cả chúng ta đã biết về Button, chúng ta click/touch lên nó và điều gì đó xảy ra trong game. Có thể chúng ta sử dụng một Button để chuyển đổi Scene, hoặc thêm vào một đối tượng Sprite trong game play. Button sẽ xử lý một callback nào đó khi chúng ta click/touch lên nó. Button cũng có hai trạng thái: **normal** và **selected**. Bề ngoài của Button có thể thay đổi theo trạng thái của nó. Chúng ta có thể tạo ra một Button gắn với một callback như sau:



Trong ví dụ này, chúng ta có thể thấy mỗi file **.png** sẽ được sử dụng để vẽ Button trong một trạng thái riêng biệt. Một Button có thể được tạo ra từ ba Sprite cho ba trạng thái khác nhau như sau:



Và Button sẽ xuất hiện trên màn hình trông như thế này:



### Checkboxs

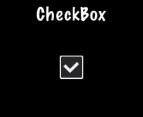
Chúng ta cũng đã từng sử dụng Checkboxs để tạo các bản khảo sát, trong trường hợp người dùng cần có nhiều lựa chọn cho một câu hỏi. Chúng ta cũng có thể sử dụng Checkboxs trong game. Có thể chúng ta sẽ sử dụng nó để người dùng đưa ra quyết định Yes hoặc No đơn giản. Checkbox có năm trạng thái khác nhau, do đó, chúng ta có thể sẽ cần đến 5 file .png khác nhau:



Với 5 trạng thái khác nhau, 5 file hình ảnh đã được sử dụng:



Đối tượng Checkbox được tạo ra có thể trông như sau:



### TextField

Trong trường hợp người chơi muốn nhập một cái tên để gắn cho nhân vật chính của game, UI widget phù hợp nhất chỉ có thể là TextField. Một TextField được sử dụng để nhập văn bản. Nó hỗ trợ các sự kiện touch, focus, percent positon và percent content size. Để tạo ra một đối tượng TextField:

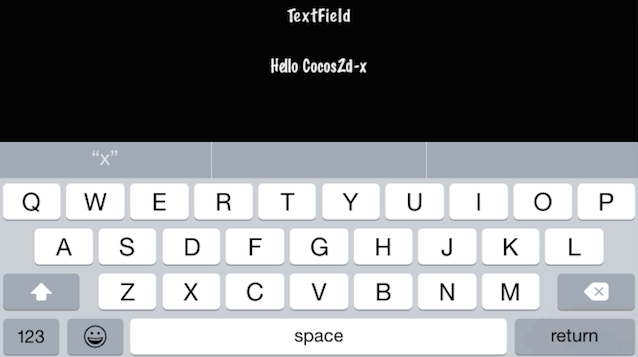
Trong ví dụ này, một TextField được tạo ra cùng với một callback.

Đối tượng TextField rất linh hoạt và có thể chứa tất cả những gì chúng ta nhập vào, và với nhiều định dạng khác nhau, ví dụ định dạng password. Chúng ta cũng có thể giới hạn số ký tự mà người dùng nhập vào:

Trên màn hình, đối tượng TextField có thể được vẽ như thế này:



Khi chúng ta click/touch vào nó, bàn phím ảo sẽ xuất hiện:



## Một số kiểu Node khác

Bên cạnh những Node cơ bản mà chúng ta đã cùng tìm hiểu trong các phần trước, Cocos2d-x còn cung cấp một số Node nâng cao dùng cho việc thiết kế những chức năng đặc biệt hơn.

### Tile Map

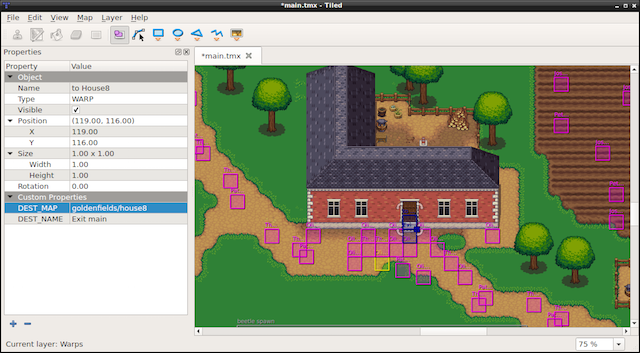
TileMaps là những map được tạo ra bởi cái Tiles, mỗi Tile có thể có hành vi độc lập. TileMaps được lưu với định dạng XML-based, gọi là TML. TMX ban đầu được thiết kế cho các tile-based map, nhưng nó cũng phù hợp để phát triển những game tổng quát hơn do nó hỗ trợ nhiều kiểu đối tượng khác nhau. Đối tượng TMX khá dễ dàng được tạo ra:

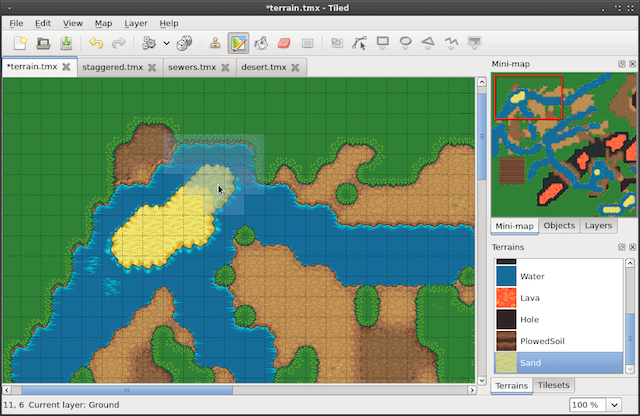
Tile-based maps có thể có nhiều Layer khác nhau, bằng cách chỉ định **z-order** cho chúng. Chúng ta có thể truy cập vào các Layer của một TMX thông qua tên của Layer:

Mỗi Tile có một ID riêng biệt, giúp chúng ta dễ dàng chọn ra một Tile bất kỳ thông qua ID của nó, nhưng trước hết, để lấy được Tile ID của một Tile, chúng ta sử dụng phương thức ***getTileGIDAt()*** như sau:



Ví dụ về một Tile Map layouts như sau:



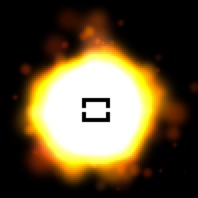


Để thiết kế được những Tile Map như thế, chúng ta có thể sử dụng **Tiled map editor,** nó khá phổ biến và được hỗ trợ bởi một cộng đồng lớn. Ảnh chụp màn hình ở trên là một ví dụ về một dự án Tiled Map.

### Particle System

Có thể chúng ta cần thêm vào game một số hiệu ứng đặc biệt như lửa cháy, vụ nổ, hay chuyển đổi kỹ năng. Lúc này, chúng ta sẽ cần đến Particle system.

Thuật ngữ Particle system đề cập đến một kỹ thuật đồ họa máy tính sử dụng một lượng lớn các đối tượng Sprite rất nhỏ để mô phỏng một loại hiện tượng mờ (**fuzzy phenomena**), mà chúng rất khó để thực hiện với những kỹ thuật dựng hình thông thường. Một số ví dụ thực tế có thể bao gồm các hệ thống hỗn độn, hiện tượng tự nhiên, phản ứng hóa học. Một số ví dụ về hiệu ứng Particle:



Chúng ta có thể sử dụng một số công cụ sau để tạo ra các hiệu ứng Particle:

- Particle Designer.

- V-Play Particle editor.

- Particle2dx.

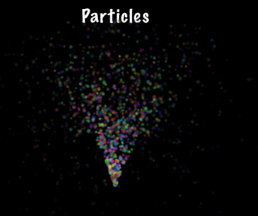
Những công cụ này thường xuất ra các file định dạng **.plist** và có thể được đọc bởi Cocos2d-x để tạo ra các đối tượng Particle trong game.



Cocos2d-x cung cấp sẵn cho chúng ta khá nhiều các kiểu Particle thông dụng:

Thử lấy một ví dụ về ParticleFireWorks như sau:

 Chúng ta thu được kết quả trên màn hình trông như thế này:

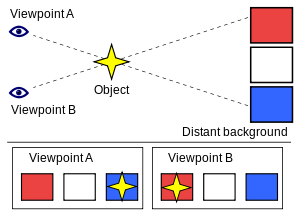


Nếu Particle được tạo ra chưa đúng với mong đợi, chúng ta hoàn toàn có thể tự mình thay đổi chúng:



### Parallax

Parallax Node là một kiểu Node đặc biệt, nó giả lập **thị sai.** Thuật ngữ "thị sai" được dùng để đặc tả cho sự thay đổi vị trí biểu kiến của một điểm trên một nền quan sát, khi nó được quan sát từ hai vị trí khác nhau, vật thể càng xa vị trí quan sát thì thị sai càng nhỏ.



Chúng ta có thể thấy hiện tượng này được mô phỏng trong một số game như Super Mario Bros. Parallax Node có thể được di chuyển xung quanh bởi một đối tượng Sequence và có thể bằng những thao tác thủ công như touch, accelerometer và keyboard events.

Parallax Node phức tạp hơn so với các Node thông thường, bởi vì nó đòi hỏi sử dụng nhiều hơn một Node để hoạt động. Một Parallax Node không thể tự hoạt động, nó đòi hỏi có ít nhất hai Node cho một hiệu ứng thị sai hoàn chỉnh. Tuy nhiên, Parallax Node rất dễ dàng tạo ra:

Chúng ta cần có thêm Node để tạo ra hiệu ứng thị sai:

Rất khó khăn để biểu diễn hiệu ứng thị sai dưới dạng văn bản, nên chúng ta chỉ có thể nhìn rõ hơn thông qua việc tự chạy đoạn code mẫu này.

## Điều phối sự kiện

### Cơ chế hoạt động của đối tượng điều phối sự kiện

Sử dụng EventDispatcher (đối tượng điều phối sự kiện) là một cơ chế hoạt động của Cocos2d-x, nó phản hồi lại những sự kiện đến từ người dùng. Cơ chế hoạt động của nó như sau:

- Trình lắng nghe sự kiện (Event listener process) đóng gói sự kiện của người dùng.

- Đối tượng điều phối sự kiện (Event dispatcher) thông báo cho Event listener về các sự kiện của người dùng.

- Những đối tượng Event chứa thông tin của sự kiện.

### Các kiểu của đối tượng lắng nghe sự kiện

Có 5 kiểu của Event listener:

* EventListenerTouch - phản hồi sự kiện touch.
* EventListenerKeyboard - phản hồi sự kiện bàn phím.
* EventListenerAcceleration - phản hồi sự kiện cảm biến gia tốc.
* EventListenerMouse - phản hồi sự kiện chuột.
* EventListenerCustom - phản hồi sự kiện tùy chỉnh.

### Độ ưu tiên của sự kiện

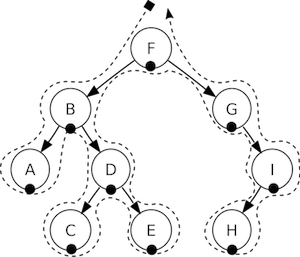
Bộ điều phối sự kiện (EventDispatcher) sử dụng khái niệm độ ưu tiên để quyết định bộ lắng nghe sự kiện nào sẽ nhận thông báo trước.

* Fixed Priority

Fixed Priority là một số nguyên được sử dụng cho mỗi đối tượng Event Listener, Fixed Priority càng lớn thì càng được ưu tiên.

* SceneGraph Priority

SceneGraph Priority là một con trỏ kiểu Node. Những EventListener Node cũng có thuộc tính **z-order,** và z-order càng lớn thì bé thì được ưu tiên hơn các EventListener có **z-order** lớn. Cơ chế hoạt động của SceneGraph Priority hoàn toàn giống với khái niệm SceneGraph đã được đề cập đến trong các phần trước.



### Xử lý các kiểu sự kiện

* Touch Events

Touch Events là event quan trọng nhất cần được xử lý trong các game mobile. Chúng dễ dàng được tạo ra và cung cấp những tính năng linh hoạt. Hãy chắc chắn rằng chúng ta biết touch events là gì. Khi chúng ta chạm vào màn hình của thiết bị mobile, nó nhận một sự kiện touch, sau đó thiết bị mobile phản hồi lại sự kiện touch vừa xảy ra. Có khả năng chúng ta không touch vào những đối tượng phản hồi, nhưng có thể có gì khác bên dưới nó. Touch Events thường được gán cho độ ưu tiên cao nhất để phản hồi cho người dùng sớm nhất có thể.

Đây là cách để tạo ra một touch event listener:

Có 3 sự kiện khác nhau được đối tượng **listener1** lắng nghe để phản hồi:

- onTouchBegan được kích hoạt khi chúng ta vừa chạm vào màn hình.

- onTouchMoved được kích hoạt khi chúng ta di chuyển vị trí touch sau khi chạm vào màn hình.

- onTouchEnded được kích hoạt khi chúng ta ngừng touch.

* Keyboard Events

Khi chơi game trên desktop, chúng ta có thể cần sử dụng đến bàn phím. Tương tự như touch events, Cocos2d-x cung cấp cho chúng ta đối tượng lắng nghe sự kiện từ bàn phím:



* Accelerometer Events

Một số thiết bị mobile được trang bị bộ cảm biến gia tốc. Một cảm biến gia tốc trên điện thoại là một thiết bị dùng để đo hướng gia tốc trọng trường. Một trường hợp thường sử dụng là khi chuyển từ chế độ **portrait** sang **landscape**. Để sử dụng bộ lắng nghe sự kiện của cảm biến gia tốc trên mobile, chúng ta cần bật chúng lên trước:

Tiếp theo, chúng ta sẽ tạo một đối tượng EventListenerAcceleration:



* Mouse Events

Cocos2d-x cũng hỗ trợ lắng nghe sự kiện của chuột:



Với 4 sự kiện khác nhau của Mouse Events, chúng ta sẽ cần 4 callback để xử lý, đầu tiên là sự kiện onMouseDown:

Một callback khác cho sự kiện onMouseUp:



Tiếp theo là callback cho sự kiện onMouseMove:

Cuối cùng là callback cho sự kiện onMouseScroll:



* Custom Events

Những kiểu sự kiện trên được định nghĩa bởi hệ thống Cocos2d-x, và chúng được kích hoạt bởi hệ thống một cách tự động. Thêm vào đó, chúng ta cũng có thể tự tạo những sự kiện tùy chỉnh cho riêng mình, chúng sẽ không được kích hoạt bởi hệ thống, mà bởi code của chúng ta, ví dụ:

Một sự kiện tùy chỉnh đã được định nghĩa như trên, với một callback method, và được thêm vào Event Dispatcher. Vậy, làm cách nào để nó được kích hoạt? Kiểm nghiệm thử với đoạn code dưới đây:

Đoạn code ví dụ trên tạo ra một đối tượng EventCustom với một tên gọi, giống với tên gọi được sử dụng cho callback method. Vậy, callback method sẽ được kích hoạt sau khi thực hiện:



### Đăng kí và xóa bỏ đối tượng xử lý sự kiện

Đăng kí một sự kiện với đối tượng Event Dispatcher khá đơn giản:

Nhưng chúng ta cần lưu ý rằng Touch Event chỉ có thể đăng ký một lần cho mỗi đối tượng. Nếu chúng ta cần nhiều hơn một listener cho nhiều đối tượng khác nhau, sử dụng phương thức ***clone()*** sẽ phù hợp cho hoàn cảnh này:

Một đối tượng Event Listener đã được đăng kí với hệ thống có thể được hủy bỏ bởi phương thức sau:



## Audio

Âm thanh là một thành phần không thể thiếu trong game. Cocos2d-x cung cấp một audio engine được gọi là **SimpleAudioEngine**. Nó có thể được sử dụng để chơi nhạc nền cũng như các hiệu ứng âm thanh trong suốt game. SimpleAudioEngine là một đối tượng **singleton**, do đó chúng ta có thể sử dụng đối tượng này bất kỳ vị trí nào trong code.

* Background music

Trong ví dụ dưới đây, Cocos2d-x sẽ được yêu cầu chơi một file nhạc như nhạc nền của game, nó có chế độ lặp lại:



* Sound effect

Và một ví dụ về chơi các hiệu ứng âm thanh:



* Pausing, stopping, resuming background music và sound effects

Sau khi bắt đầu chơi một file âm thanh nào đó, chúng ta có thể cần chúng tạm dừng tai một thời điểm nào đó. Điều này có thể thực hiện khá dễ dàng.

- Pause



- Stop



- Resume



* Một số tính năng nâng cao khi sử dụng Cocos2d-x Audio Engine

- Setup

Với những API đơn giản của SimpleAudioEngine của Cocos2d-x cung cấp, chúng ta có thể bắt đầu sử dụng chúng một cách dễ dàng. Tuy nhiên, có một số cân nhắc cần nhớ khi sử dụng chúng trong game, nhất là khi game được thực thi trên các hệ điều hành của điện thoại và tablets. Điều gì xảy ra nếu chúng ta chuyển đổi qua lại giữa các ứng dụng? Hoặc khi có một cuộc gọi đến bất ngờ? Chúng ta có thể xử lý những vấn đề này trong phần điều khiển chu kỳ hoạt động của game ứng với các hệ điều hành khác nhau. Một điều đặc biệt là Cocos2d-x đã cung cấp cho chúng ta những API có thể xử lý trên nhiều hệ điều hành khác nhau. Về phần xử lý chu kỳ sống của ứng dụng game, chúng ta thực hiện trong file **AppDelegate.cpp:**



- Preload sound

Khi một game được bắt đầu, chúng ta có thể muốn tải trước nội dung các file âm thanh lên nhằm mục đích tối ưu hóa trong phần game play.



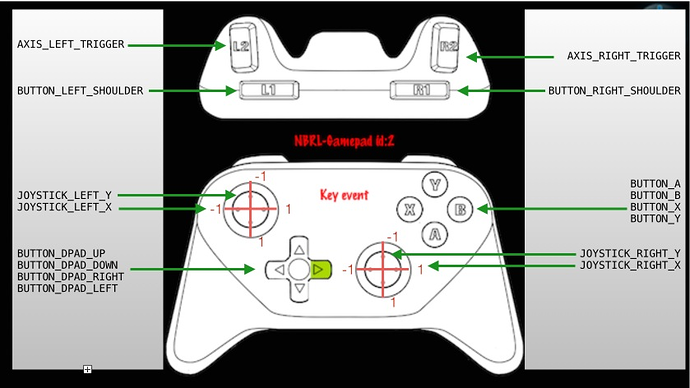
- Volume

SimpleAudioEngine cung cấp cho chúng ta những API để thay đổi âm lượng:



## Game controller

Game Controller là một kiểu Event Listener đặc biệt trong Cocos2d-x, được gọi là **EventListenerController**. Có 6 sự kiện cần lắng nghe từ controller: *onConnected*, *onDisconnected*, *onKeyDown*, *onKeyUp*, *onAxisEvent*, *onKeyRepeat*. Sự kiện onAxisEvent là sự kiện xảy ra khi thanh analog thay đổi. *keyCode* đề cập đến phím trên controller được tương tác. Dưới đây là danh sách các *keyCode* và nó được phản hồi cho các phím tương ứng:

Có 3 trạng thái chính của mỗi phím. Nếu *isAnalog* là true, giá trị có thể từ -1 đến 1. Nếu *isAnalog* là false, giá trị có thể bị giới hạn từ 0 đến 1. Điều này phụ thuộc vào loại controller đang sử dụng. Lấy ví dụ, nếu key là JOYSTICK\_LEFT\_X, và trường *isAnalog* của nó là true, giá trị của nó sẽ nằm trong giới hạn từ -1 đến 1, nó biểu diễn trục Ox cho joystick đang được sử dụng.



Và ví dụ dưới đây cho chúng ta thấy cách sử dụng những sự kiện của controller trong game như thế nào:

Đầu tiên, chúng ta có thể tạo một phương thức để đăng kí EventListenerController với hệ thống, gắn các callback function cho nó.

Tiếp theo là xử lý cho các callback functions, và cuối cùng là gọi hàm đăng ký.



## Physics

Với những khái niệm chúng ta đã tìm hiểu trong các phần trước, có thể đã đủ cho chúng ta thiết kế một số game đơn giản. Chúng ta có thể tạo ra những đối tượng Sprite, thiết kế game play với các Action. Nhưng chúng ta sẽ làm gì nếu cần mô phỏng các tính chất vật lý trong game? Các tính chất vật lý có thể bao gồm: *hiện tượng va chạm*, *trọng lực*, *ma sát*, *đàn hồi*, … Trong phần này, chúng ta sẽ cùng tìm hiểu khi nào cần áp dụng Physics và sử dụng Physics Engine.

### Physics engine - có cần thiết hay không?

Khái niệm Physics (Vật lý) có thể rất kinh khủng đối với nhiều người. Đó là lý do chúng ta cần có một Physics Engine. Có thể chúng ta chỉ cần thiết kế đơn giản với những đối tượng Node, sử dụng đa hình cho phương thức ***update()*** kết hợp với các đối tượng Rect như sau:

Cơ chế này hoạt động khá đơn giản, nhưng không có khả năng mở rộng. Thử tưởng tượng chúng ta có khoảng hơn 100 đối tượng Sprite, và tất cả chúng cùng gọi đến phương thức ***update()*** để kiểm tra sự giao nhau? Điều này hoàn toàn có thể, nhưng tốn khá nhiều chi phí để CPU có thể thực thi, và có thể gây giảm framerate, trường hợp xấu hơn nữa là game không thể chơi được. Physics Engin được tạo ra để giải quyết những vẫn đề này cho chúng ta, với khả năng mở rộng cao và sử dụng CPU một cách hợp lý. Chúng ta có thể bắt đầu tìm hiểu Physics Engine bằng một ví dụ:

Đây là những gì mà đoạn code trên thực hiện, từng bước một: một đối tượng PhysicsBody được tạo ra, một đối tượng Sprite được tạo ra, các tính chất vật lý của đối tượng PhysicsBody được áp dụng lên đối tượng Sprite. Sau đó, một đối tượng lắng nghe sự kiện va chạm được tạo ra, callback ***onContactBegin()*** sẽ phản hồi lại khi có những va chạm xảy ra trong Scene.

### Những thuật ngữ và khái niệm thường dùng

* Bodies

Một Physics Body nắm giữ các thuộc tính vật lý của một đối tượng. Nó bao gồm khối lượng, vị trí, sự quay, vận tốc, hệ số giảm sóc… Những đối tượng Physics Body là xương sống của các Shape. Một Physics Body không có một hình thể nhất định trừ khi được gắn vào một đối tượng Shape.

* Material

Material mô tả các tính chất:

- Tỉ trọng (density): được sử dụng để tính toán khối lượng của một Body.

- Ma sát (friction): làm cho vật thể trượt một cách thực tế hơn.

- Đàn hồi (restituation): làm cho đối tượng bật ngược trở lại. Giá trị hệ số đàn hồi thường được thiết lập trong khoảng từ 0 đến 1, với 1 có nghĩa vật thể sẽ bật ngược với vector có cùng độ lớn lúc va chạm.

* Shapes

Đối tượng Shape miêu tả va chạm hình học. Bằng cách gắn Shape vào Body, chúng ta định nghĩa được một Body Shape. Chúng ta có thể gắn nhiều Shape cho một Body nếu chúng ta cần định nghĩa một Shape phức tạp. Mỗi Shape liên quan đến một đối tượng PhysicsMaterial chứa các thuộc tính *type*, *area*, *mass*, *moment*, *offset* và *tag*.

- *type*: mô tả các loại hình dạng của Shape, như circle, box, polygon…

- *area*: dùng để tính toán thuộc tính *mass* của Body. Từ thuộc tính *density* và *area* có thể suy ra *mass*.

- *mass*: đặc trưng cho số lượng vật chất được chứa trong Body, được tính bởi gia tốc của nó dưới tác động của một lực hoặc bởi lực tác dụng lên nó bởi một trường hấp dẫn.

- *moment*: xác định mo-men quay cần thiết cho một gia tốc quay.

- *offset*: phần bù tính từ trọng tâm của Body.

- *tag*: dùng để xác định Shape dễ dàng hơn cho các nhà phát triển. Tất cả các Node đều có thể được gắn tag.

Cocos2d-x mô tả các kiểu khác nhau của Shape như sau:

- PhysicsShape: là lớp được sử dụng để kế thừa cho các kiểu Shape khác.

- PhysicsShapeCircle: là kiểu Shape hình tròn đặc.

- PhysicsShapePolygon: là kiểu Shape đa giác đặc.

- PhysicsShapeBox: là kiểu Shape hình chữ nhật.

- PhysicsShapeEdgeSegment: một phân đoạn của Shape.

- PhysicsShapeEdgePolygon: là kiểu hình đa giác rỗng.

- PhysicsShapeEdgeBox: là kiểu hình chữ nhật rỗng.

- PhysicsShapeEdgeChain: kiểu dây chuyền, cung cấp khả năng kết nối các cạnh một cách hiệu quả.

* Contacts / Joints

Contacts và Joints mô tả cách mà các Body được gắn kết với nhau.

* World

Một World là một không gian mà trong đó được giả lập tính chất vật lý. Khi thêm các đối tượng có Physics Body vào không gian này, chúng sẽ chịu ảnh hưởng, tương tác với nhau bởi các định luật vật lý.

### Physics world và Physics body

* Physics World

Một đối tượng PhysicsWorld là cốt lõi khi giả lập vật lý. Cũng giống như thế giới chúng ta đang sống, một PhysicsWorld có nhiều thứ xảy ra cùng lúc. PhysicsWorld được tích hợp sâu bên trong những Scene bởi có nhiều khía cạnh. Dưới đây là một ví dụ tạo ra một Scene được tích hợp PhysicsWorld:

Tất cả đối tượng PhysicsWorld đều có những thuộc tính:

- gravity: trọng lực toàn cầu áp dụng cho PhysicsWorld, mặc định là Vec2(0.0F, -98.0F).

- speed: là một tỉ lệ mô phỏng thực thi tính chất vật lý, mặc định là 1.0.

- updateRate: là một tỉ lệ thực hiện update Physics, tỉ lệ update được tính bởi *(EngineUpdateTimes / PhysicsWorldUpdateTimes)*.

- substeps: dùng để thiết lập số lượng substep của việc update physics.

* Physics Body

Đối tượng PhysicsBody có những thuộc tính như **position** và **velocity**. Chúng ta có thể áp dụng các lực (*forces*), động lượng (*movement*), giảm sóc (*damping*), xung (*impulse*). PhysicsBody có thể ở trong trạng thái tĩnh (*static*) hoặc động (*dynamic*). Một PhysicsBody tĩnh không di chuyển dưới tác động của các tính chất vật lý, và ngược lại. Đoạn code dưới đây cho thấy kết quả khác biệt giữa việc áp dụng PhysicsBody tĩnh và PhysicsBody động vào trong một Scene:



### Collision

Có thể chúng ta đã từng biết về các vụ tai nạn oto, lúc đó, hai Shape của hai chiếc oto giao nhau trong không gian tạo ra sự va chạm. Collision là những gì xảy ra khi những đối tượng PhysicsBody va chạm vào nhau.

Trong Cocos2d-x, chúng ta có thể bỏ qua việc xử lý một số cặp đối tượng va chạm vào nhau, bằng cách sử dụng Filtering Collision. Physics Engine hỗ trợ filtering collision sử dụng **category** và **group bitmask**.

Với mỗi Shape, chúng ta có thể chỉ định một giá trị *category*, chúng ta cũng có thể chỉ định những category nào sẽ xảy ra va chạm với nhau. Điều này có thể được thực hiện bởi bitmask bit:

Chúng ta có thể kiểm tra va chạm bằng cách kiểm tra *category* và *collision bitmask*:

 Colilsion group cho phép chúng ta tách các nhóm va chạm ra. Va chạm giữa các Shape nằm trong các nhóm khác nhau được lọc ra bởi *collision bitmask* và *category*. Chú ý rằng lọc theo nhóm có độ ưu tiên cao hơn lọc theo bitmask.

### Collision detection

Các sự va chạm giữa các đối tượng được quản lý bởi Physics Engine. Một đối tượng Contact sẽ không được tạo ra bởi người dùng, chúng được tạo ra một cách tự động. Có một số thuật ngữ được gắn liền với sự va chạm:

- contact point: là điểm mà hai Shape chạm vào nhau.

- contact normal: là vector đơn vị hướng từ một Shape đến một Shape khác khi xảy ra va chạm.

Chúng ta có thể lấy được các cặp PhysicsShape từ một sự va chạm:



Chúng ta có thể truy cập những đối tượng Contact bằng cách thực hiện một Contact Listener. Contact Listener hỗ trợ một số sự kiện va chạm như: *begin*, *pre-solve*, *post-solve*, *separate*.

- *begin*: hai đối tượng Shape chỉ vừa mới va chạm vào nhau. Return true từ callback này sẽ xử lý va chạm một cách bình thường, ngược lại, return false từ đây sẽ làm Physics Engine bỏ qua việc xử lý các trạng thái va chạm còn lại.

- *pre-solve*: hai đối tượng Shape đang va chạm với nhau.

- *post-solve*: hai đối tượng Shape đang va chạm với nhau và việc phản hồi va chạm đã được xử lý.

- *separate*: hai đối tượng Shape ngừng va chạm vào nhau.

Dưới đây là một ví dụ về việc xử lý va chạm bằng cách tạo một đối tượng EventListenerPhysicsContact và liên kết các sự kiện với những callback khác nhau:



# PHÂN TÍCH, THIẾT KẾ GAME BEWARE OF DUNGEON

## Giới thiệu game

Game được xây dựng theo thể loại game trí tuệ, không tập trung quá nhiều vào đồ họa, mà chú trọng vào việc đặt ra những thử thách khó đòi hỏi người chơi phải vượt qua với sự suy đoán, ước lượng của mình.

### Cốt truyện của game

Trong một đêm nọ, Masha (tên của nhân vật chính) đang ngủ say và có một giấc mơ đẹp. Một kẻ biến thái lặng lẽ đột nhập vào phòng ngủ và bắt cóc cô ấy. Khi tỉnh giấc, Masha nhận ra mình đã bị nhốt trong một căn phòng, phía sâu trong cùng của một ngục tối. Tuy nhiên, cửa phòng không bị khóa, cô ấy bước ra bên ngoài và nhận ra xung quanh mình là những con quái vật kì dị chưa từng được biết đến. Điều duy nhất Masha nghĩ tới vào lúc này là tìm một đường đi để thoát ra khỏi chổ tối tăm này, trước khi điều gì đó khủng khiếp hơn xảy đến với mình. Nhưng Masha quá hoảng sợ nên cô ấy không tìm ra được cách băng qua bọn quái vật này, nên cô ấy trông chờ vào bạn. Hãy giúp Masha chọn một đường đi chính xác, và một thời điểm thích hợp để chạy trên con đường bạn đã vạch ra, cố gắng đừng để Masha đi vào trong tầm với của lũ quái vật. Sau khi vượt qua tất cả các phòng giam, cô ấy có thể trở về nơi an toàn.

### Các quy luật trong game

* Di chuyển

Mỗi màn chơi đều được thiết kế bằng Tile Map Editor, với kích thước mỗi tile là **64x64 pixel**. Tất cả quái vật, và nhân vật chính chỉ có thể di chuyển từ tâm của tile này sang tâm của tile khác trong một khoảng thời gian nhất định (Ngoại trừ loại quái vật có khả năng săn đuổi).

Tốc độ di chuyển của nhân vật chính bị cố định ở giá trị **2.0F**, và tốc độ di chuyển của các quái vật sẽ nằm trong khoảng từ **1.0F** đến **2.0F.** Tốc độ di chuyển của quái vật sẽ không nhanh hơn nhân vật chính.

Một khi nhân vật chính bắt đầu di chuyển, nó sẽ không dừng lại cho đến khi đạt đến cánh cửa kết thúc màn chơi.

Tốc độ di chuyển của quái vật và nhân vật chính phải được hiển thị trên bảng hướng dẫn.

Nếu nhân vật chính di chuyển vào phạm vi tấn công của bất kỳ quái vật nào, nhân vật chính sẽ bị bắt. Trong lượt chơi tiếp theo, nhân vật chính phải trở lại vị trí xuất phát.

Khi nhân vật chính dừng lại tại cánh cửa phía bên kia, nhân vật chính sẽ không bị quái vật bắt nữa cho dù nhân vật chính nằm trong phạm vi tấn công của quái vật.

* Mở các màn chơi

Màn chơi tiếp theo chỉ được mở khi vào chỉ khi người chơi đến được với cánh cửa phía bên kia trong căn phòng hiện tại.

Nếu người chơi thoát khỏi game, hoặc thoát khỏi màn chơi hiện tại bằng cách nhấn button "***Give Up***", màn chơi đó sẽ không được tính.

Người chơi có thể chơi lại màn chơi đã vượt qua được trước đó, nhưng màn chơi mới vẫn không được mở cho đến khi vượt qua được màn chơi cuối cùng trong số các màn chơi đã được mở.

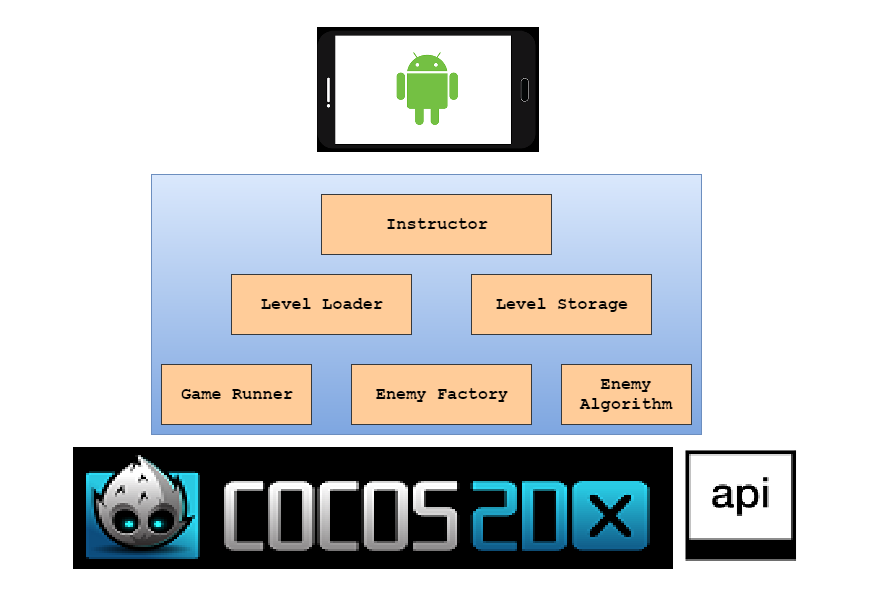
* Điều kiện thắng

Nhân vật chính chỉ được giải phóng sau khi thoát ra được khỏi căn phòng cuối cùng gần lối ra nhất.

Sau khi thắng, các màn chơi vẫn được mở sẵn và người chơi có thể chơi lại bất kỳ màn chơi nào trong số tất cả màn chơi.

## Kiến trúc của game

Dưới đây là kiến trúc tổng thế của game được xây dựng trên nền tảng Cocos2d-x.



Các thành phần chính trong kiến trúc này bao gồm:

* Game Scene

Là một Scene được sử dụng chung cho tất cả màn chơi. Với mỗi màn chơi được tải lên, chúng sẽ được xử lý trong Game Scene, cùng với đó là tất cả những đối tượng khác đều hoạt động bên trong Scene này.

* Enemy Factory

Nhằm phục vụ cho việc tạo ra các đối tượng quái vật có kiểu khác nhau. Mỗi đối tượng quái vật được xác định bởi các thuộc tính trong Tile Map.

* Enemy Algorithm

Mỗi loại quái vật sẽ có một chu kỳ di chuyển khác nhau, thuật toán di chuyển cho từng loại quái vật đóng vai trò quan trọng trong kiến trúc của game.

* Level Loader

Được sử dụng để tải các màn chơi.

* Level Storage

Các màn chơi sẽ được lưu trữ và quản lý với thành phần này.

* Instructor

Bất kỳ game nào cũng cần có phần hướng dẫn giúp người chơi làm quen với các thao tác trong game.

## Các tính năng của game

### Các tính năng chính

Dưới đây là một số tính năng đáng chú ý trong game:

* Hệ thống chuyển đổi Scene

Quản lý dòng chuyển đổi giữa các Scene trong game.

* Hệ thống bản đồ, các màn chơi

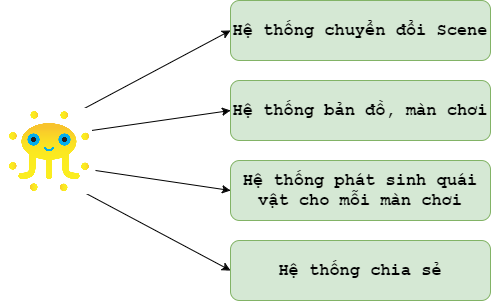
Quản lý các màn chơi trong game.

* Hệ thống phát sinh quái vật cho mỗi màn chơi

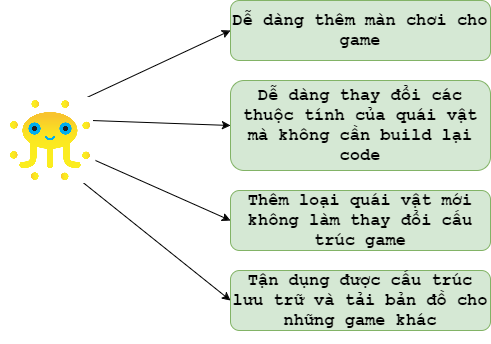
Phân loại quái vật dựa trên kiểu của chúng để tạo ra những đối tượng quái vật có cách di chuyển tương ứng.

* Chia sẻ game qua mạng xã hội

Tích hợp chia sẻ thông tin game qua mạng xã hội Facebook.



### Một số tính năng mở rộng



## Phân tích, thiết kế game

### Hệ thống chuyển đổi Scene

Game "*Beware of dungeon*" được thiết kế bao gồm những Scene dưới đây:

* Splash Scene

Dùng để hiển thị logo của phía cung cấp game engine.

* Main Menu Scene

Dùng làm điểm dừng để người chơi chuẩn bị trước khi vào game.

* Level Selection Scene

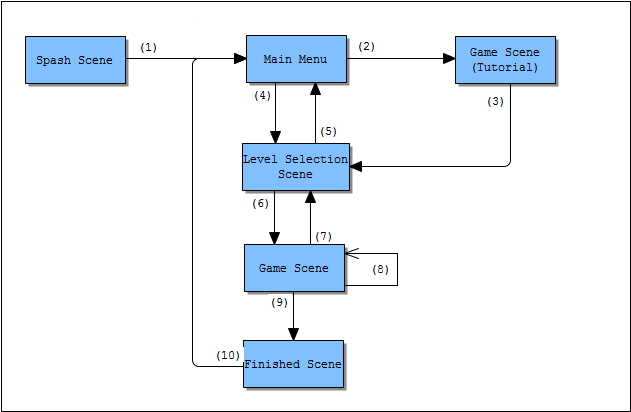
Dùng để tải các bản đồ và trạng thái của các màn chơi trong game, và đảm nhiệm chức năng chuyển hướng vào Game Scene.

* Game Scene

Là nơi diễn ra các hoạt động của một màn chơi.

* Finished Scene

Dùng để chúc mừng người chơi đã vượt qua thử thách, và hiển thị thông tin những người tham gia vào quá trình phát triển game.



Chú thích dòng chuyển đổi Scene:

(1) Quá trình tự động chuyển từ Splash Scene sang Main Menu Scene sau khi hiển thị logo của phía cung cấp game engine sau hai giây.

(2) Chuyển từ Main Menu Scene vào thẳng Game Scene sau khi nhấn button Play nếu đây là lần đầu tiên chơi game, màn chơi đơn giản và có hướng dẫn sẽ được tải lên.

(3) Chuyển từ Game Scene về Level Selection Scene sau khi người chơi đã hoàn thành màn chơi có hướng dẫn.

(4) Từ Main Menu Scene, người chơi nhấn button Play. Nếu người chơi đã hoàn thành màn chơi có hướng dẫn, họ sẽ được chuyển thẳng đến Level Selection Scene để lựa chọn màn chơi.

(5) Từ Level Selection Scene, người chơi nhấn button Back (hoặc Escape nếu sử dụng bàn phím) và trở về Main Menu Scene.

(6) Từ Level Selection Scene, người chơi chọn một màn chơi và được chuyển đến Game Scene.

(7) Trong lúc chơi game, người chơi có thể nhấn button "*Give Up*" để trở lại Level Selection Scene.

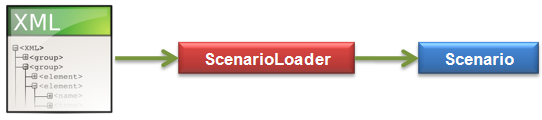
(8) Là quá trình tự động tải lại màn chơi hiện tại nếu nhân vật chính trong game bị bắt.

(9) Sau khi vượt qua được màn chơi cuối cùng, người chơi sẽ được chuyển đến Finished Scene.

(10) Là quá trình tự động chuyển về Main Menu Scene sau khi thực hiện xong chuỗi hành động. Người chơi có thể ép chuyển về Main Menu Scene ngay bằng cách nhấn button Back.

### Hệ thống bản đồ, màn chơi

- Hình ảnh tương tự



- File đặc tả màn chơi cần có những thông tin

- biểu đồ hoạt động init load màn chơi

- biểu đồ hoạt động kết thúc màn chơi

- biểu đồ tuần tự init load màn chơi

- biểu đồ tuần tự kết thúc màn chơi

- biểu đồ lớp các lớp tham gia vào hệ thống bản đồ, màn chơi

### Hệ thống phát sinh quái vật cho mỗi màn chơi

### Hệ thống chia sẻ

# TRIỂN KHAI VÀ ĐÁNH GIÁ KẾT QUẢ

# DANH MỤC TÀI LIỆU THAM KHẢO

1. SlackMoehrle, Ricardo, Justin, Nite, Kai, Minggo, Wenhai, Tony, Yingtao, Rao, Huabin, Zhe. “Cocos2d-x Programmers Guide”  
   <http://www.cocos2d-x.org/docs/static-pages/programmers-guide.html>
2. Roger Engelbert, “Cocos2d-x by Example Beginner's Guide”, April 2013.
3. Sonar system, “Cocos2d-x v3 Course”  
   <http://www.sonarlearning.co.uk/coursepage.php?topic=game&course=cocos2d-x-v3>