|  |
| --- |
|  |

**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**ĐỒ ÁN TỐT NGHIỆP**

**Xây dựng Engine Game**

**Nguyễn Thành Bắc**

bac.nt150264@sis.hust.edu.vn

**Ngành Công nghệ thông tin và truyền thông**

**Chuyên ngành Công nghệ phần mềm**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn:** | ThS. Lê Tấn Hùng |
| **Bộ môn:** | Công nghệ phần mềm |
| **Viện:** | Công nghệ thông tin – Truyền thông |
| **HÀ NỘI, 06/2020** | |

**Lời cam kết**

Họ và tên sinh viên: . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

Điện thoại liên lạc: . . . . . . . . . . . . . . . . . . . . . Email: . . . . . . . . . . . . . . . . . . . . . . . .

Lớp: . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . Hệ đào tạo: . . . . . . . . . . . . . . . . . . . .

Tôi – Nguyễn Thành Bắc – cam kết Đồ án Tốt nghiệp (ĐATN) là công trình nghiên cứu của bản thân tôi dưới sự hướng dẫn của ThS. Lê Tấn Hùng. Các kết quả nêu trong ĐATN là trung thực, là thành quả của riêng tôi, không sao chép theo bất kỳ công trình nào khác. Tất cả những tham khảo trong ĐATN – bao gồm hình ảnh, bảng biểu, số liệu, và các câu từ trích dẫn – đều được ghi rõ ràng và đầy đủ nguồn gốc trong danh mục tài liệu tham khảo. Tôi xin hoàn toàn chịu trách nhiệm với dù chỉ một sao chép vi phạm quy chế của nhà trường.

|  |  |
| --- | --- |
|  | *Hà Nội, ngày tháng năm*  Tác giả ĐATN  *Họ và tên sinh viên* |

**LỜI MỞ ĐẦU**

Đối với mỗi sinh viên trường đại học bách khoa Hà Nội, để có thể đạt được những thành công trong con đường sự nghiệp của mình thì vượt qua đồ án tốt nghiệp là một cột mốc quan trọng, làm bàn đạp vững chắc để đối mặt với những khó khăn, thách thức sau này. Tuy nhiên để chinh phục được thử thách này không dễ dàng đạt được mà dựa vào sự khổ luyện của mỗi người, và đặc biệt là sự hỗ trợ đến từ thầy cô, bạn bè và người thân.

Lời đầu tiên em xin gửi lời cảm ơn đến ban giám hiệu nhà trường và công ty TNHH Gameloft Việt Nam đã tạo điều kiện cho em có một môi trường học tập, làm việc cũng như phát triển các kỹ năng của bản thân.

Tiếp theo em xin gửi lời cảm ơn đến thầy giáo hướng dẫn là thầy Lê Tấn Hùng, thầy là một người nhiệt huyết với nghề, không ngại thời gian, công việc để hỗ trợ, chỉ bảo em hoàn thành đồ án một cách tốt nhất.

Sau đó em xin gửi lời cảm ơn đến anh Hoàng Mạnh Hưng – Technical Leader của công ty GameLoft người đã có kinh nghiệm lâu năm trong việc code engine game. Anh Hưng là người đồng hành và giúp đỡ em trong suốt quá trình tìm hiểu, nghiên cứu và hoàn thành đồ án.

Cảm ơn bạn bè, những người luôn bên em những lúc khó khăn và chán nản. Em xin gửi lời cảm ơn đến bạn Phan Quang Dũng, Phí Văn Tuấn, Nguyễn Văn Quyết và Vũ Đức Hiệp là những người đã giúp em tìm hiểu và nghiên cứu hệ thống.

Sau cùng em xin gửi lời biết ơn sâu sắc đến những người thân yêu đã luôn đồng hành và làm chỗ dựa vững chắc cho em trong suốt quá trình học tập tại trường đại học bách khoa Hà Nội. Mặc dù em đã cố gắng hoàn thành báo cáo trong phạm vi và khả năng cho phép, nhưng chắc chắn sẽ không tránh khỏi những thiếu sót, kính mong sự cảm thông và tận tình chỉ bảo của quý thầy cô và các bạn.

**TÓM TẮT BÁO CÁO**

Đồ án của em là xây dựng engine game dựa trên mã nguồn mở đã cung cấp sẵn các thư viện cần thiết là SDL2 và sử dụng mẫu ECS thay thế cho Unity Engine. Chủ đề của em là tự viết ra engine game riêng để tối ưu hóa hiệu năng của trò chơi cũng như kích cỡ file build thay vì sử dụng engine có sẵn từ bên ngoài. Từ những vấn đề đã gặp phải khi xây dựng trò chơi dựa vào Unity Engine như game chơi bị giật cục, crash game, hay game không thể chạy được trên các máy đời thấp. Trong thời đại số như hiện nay, trò chơi điện tử ngày càng phát triển để đáp ứng nhu cầu về giải trí và giải tỏa căng thẳng. Thậm chí ở một vài nước, khu vực trên thế giới đã công nhận trò chơi điện tử là một trong các bộn môn thể thao trong các thế vận hội như liên quân mobile trong Seagame 30. Vì sự tăng trường mạnh mẽ như vậy mà nhà phát triển game phải không ngừng cải thiện trò chơi để mang đến những trải nghiệm tốt nhất cho người dùng. Cụ thể ở đây là việc khắc phục được những vấn đề em đã nêu ở trên, vì vậy em lựa chọn xây dựng một engine riêng để có thể giảm bớt các thư viện, tính năng hay các nền tảng mà Unity Engine đính kèm nhằm cải thiện kích thước của engine, đồng thời giảm kích cỡ file build và tăng hiệu năng của game.

Hướng tiếp cận của em là xây dựng các component module với các nghiệp vụ cơ bản như xử lý va chạm, tải ảnh, xử lý sự kiện, và các thay đổi về chuyển động của các đối tượng trò chơi. Từ đó bất kỳ một đối tượng trò chơi nào cũng có thể tái sử dụng các component tùy vào mục đích của mỗi đối tượng được sinh ra.

**Mục lục**

**Chương 1 Giới thiệu về đề tài** 7

1.1 Đặt vấn đề 7

1.2 Mục tiêu và phạm vi của đề tài 8

1.2.1 Mục tiêu của đề tài 8

1.2.2 Phạm vi đề tài 8

1.3 Định hướng giải pháp 8

1.3.1 Mô tả bài toán 8

1.3.2 Định hướng 9

**Chương 2 Tổng quan về công nghệ và kỹ thuật** 10

2.1 Giới thiệu 10

2.2 Hạn chế của kế thừa OOP 11

2.2.1 Mô hình cứng nhắc 12

2.2.2 Object-Oriented Composition 13

2.2 ECS 14

2.2.1 Lịch sử 14

2.2.2 Kiến trúc 14

2.2.3 Entity 15

2.2.4 Component 15

2.2.5 System 16

2.3 Composing the model. 17

2.4 Thiết kế hướng dữ liệu 18

2.5 Phân tích hệ thống ECS 19

2.6 Cupcake ECS 21

2.6.1 Architecture 22

2.6.2 Cupcake Solutions 22

2.7 ARTEMIS ECS FRAMEWORK 23

2.7.1 Architecture 23

2.7.2 Analysis 24

2.7.3 Các dự án sử dụng ECS 25

2.8 Lợi ích của ECS 26

2.9 Giới thiệu về SDL 27

2.10 Lịch sử và kiến trúc 28

2.10.1 Lịch sử 28

2.10.2 Kiến trúc 29

2.11 Các khái niệm cơ bản trong SDL 30

2.11.1 Tile 30

2.11.2 Sprite sheet 31

2.11.3 Frame 31

2.11.4 Map 32

2.11.5 Surface 32

2.11.6 Structure Containers 33

2.12 Các vấn đề gặp phải trong quá trình sử dụng SDL 33

2.12.1 Khởi tạo cửa sổ 33

2.12.2 LoadTexture 36

2.12.3 Kỹ thuật Color Key 36

2.12.4 Sự kiện trong SDL 37

2.12.5 Bắt sự kiện nhấn hoặc thả ra. 38

2.12.6 Xử lý di chuyển cho đối tượng nhân vật 39

2.12.7 Xây dựng lớp Vector2D 40

2.12.8 Update, Render, Clean 41

2.12.9 Play Audio trong SDL 42

**Chương 3 Xây dưng engine và phát triển game Flappy Bird** 45

3.1 Xây dựng engine game sử dụng mã nguồn mở SDL2. 45

3.1.1 Các thành phần cấu tạo có trong engine game. 45

3.1.2 Phát triển engine 45

3.1.2.1 Component 45

3.1.2.2 Entity 46

3.1.2.3 Manager 47

3.1.2.4 TransformComponent 48

3.1.2.5 SpriteComponent 49

3.1.2.6 KeyboardComponent 49

3.1.2.7 ColliderComponent 50

3.2 Phát triển game bằng engine tự xây dựng và Unity Engine 55

3.2.1 Giới thiệu về game Flapy Bird 55

3.2.2 Các đối tượng trong game. 55

3.2.3 Phát triển game bằng engine tự xây dựng. 58

3.2.4 Phát triển game bằng unity engine 59

3.2.4.1 Các thành phần có trong Unity Editor 59

3.2.4.2 Xây dựng game FlappyBird 63

**Chương 4 Kết quả đạt được** 66

4.1 Kết quả 66

4.1.1 Giao diện Menu Game 66

4.1.2 Giao diện Game Play 66

4.1.3 Giao diện Game Over 67

4.2 So sánh các chỉ số của 2 engine 67

4.2.1 So sánh size build 68

4.2.1.1 Size build game với Unity Engine 68

4.2.1.2 Size build với Engine tự xây dựng 68

4.2.1.3 Đánh giá 69

4.3 KẾT LUẬN 70

TÀI LIỆU THAM KHẢO 71

Chương 1 Giới thiệu về đề tài

1.1 Đặt vấn đề

Với sự phát triển mạnh mẽ của mạng Internet, và công nghệ thời đại 4.0, con người càng tiếp cận nhiều đến các thiết bị thông minh như điện thoại, máy tính, laptop, cùng với đó là các nhu cầu về liên lạc, công việc hay giải trí thì trò chơi điện tử cũng là một trong những thị trường phát triển mạnh mẽ nhất với các thể loại trò chơi trên máy tính hay trò chơi trên điện thoại. Tuy nhiên do sự phát triển mạnh mẽ vậy, người dùng càng ngày càng khó tính trong việc trải nghiệm game.

Cụ thể là trong dự án game nhóm em đang phát triển dựa trên Unity Engine. Ban đầu dự án ra sản phẩm khá tốt, game chơi rất mượt và có thể chạy trên các thiết bị di động kể cả các máy đời thấp. Tuy nhiên, sau một thời gian mở rộng game, dự án có thêm khá nhiều các thư viện, plug in dẫn đến việc kích cỡ file build tăng cao, đối với các máy đời thấp game không chạy được hoặc chạy được nhưng không mượt, thường bị giật cục hay có thể bị crash game do tràn ram. Vì lý do này em quyết định xây dựng một engine riêng sử dụng mẫu ECS và mã nguồn mở SDL2 nhằm giảm bớt các thư viện không cần thiết mà Unity Engine đính kèm và tối ưu hiệu năng của game.

1.2 Mục tiêu và phạm vi của đề tài

Với giới hạn về thời gian thực hiện đồ án thì game engine em tự build sẽ phát triển cho dạng game casual dễ làm để phục vụ cho việc demo sản phẩm và các module xây dựng sẽ dựa trên ECS Pattern.

1.2.1 Mục tiêu của đề tài

Thể hiện được ưu điểm của việc tự xây một engine game riêng hơn là dùng một engine đã có sẵn.

Thể hiện được tổng quan các bước xây dựng các module game.

1.2.2 Phạm vi đề tài

Xây dựng engine phát triển game Flappy Bird dựa vào mã nguồn mở SDL2.

So sánh hiệu năng và size của file build trên engine tự xây dựng với UnityEngine

1.3 Định hướng giải pháp

1.3.1 Mô tả bài toán

Xây dựng một bộ công cụ cơ bản để phát triển và có thể tái sử dụng từng thành phần trong đó. Các thành phần cơ bản mà game engine phải có là công cụ vật lý (hay công cụ tính toán và phát hiện va chạm), âm thanh, mã nguồn hay các hình ảnh động.

Các thành phần cơ bản bao gồm:

* Game Object
* Component
* Transform
* Collision
* KeyboardController
* Sprite Component
* Sound & Music

1.3.2 Định hướng

Sử dụng ECS (Entity – Component – System).

Là một mẫu cho phép nhà phát triển áp dụng để cấu hình, thay đổi hành vi cho đối tượng trò chơi bằng cách thêm bớt các module nhỏ rồi cấu hình cho từng module nhỏ đó. Ví dụ:

Muốn nhân vật bị trúng đạn và mất máu ta cần thêm Collision Component (thành phần xử lý va chạm) và Health Component (thành phần xử lý máu của nhân vật), sau đó config các thông số cho các component.

Muốn nhân vật bị bất động khi trúng đạn ta cần xóa Movable Component (thành phần di chuyển của nhân vật) đi.

Có thể tạo ra những thuộc tính phức tạp dựa vào các module nhỏ đã xây dựng trước đó.

Sử dụng mã nguồn mở SDL2

SDL cung cấp các thư viện cần thiết cho việc xây dung engine game như joystick, âm thanh, hình ảnh…

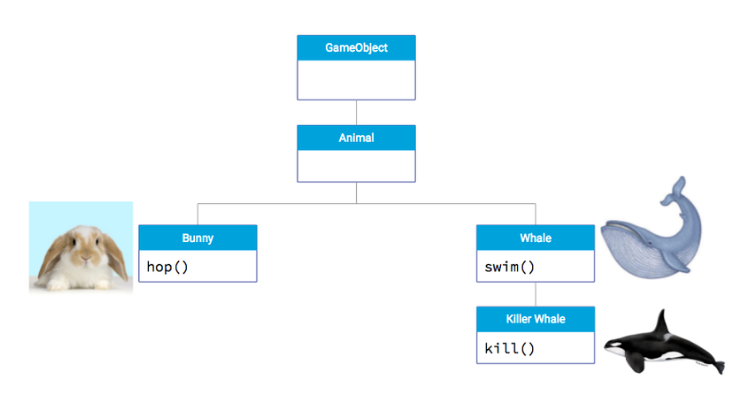
Chương 2 Tổng quan về công nghệ và kỹ thuật

2.1 Giới thiệu

Cố gắng thực hiện các ứng dụng phức tạp bằng kiến ​​trúc lập trình hướng đối tượng truyền thống thường có thể là một nhiệm vụ rất khó khăn. Các hệ thống phân cấp lớp lớn, xuất phát từ cùng một lớp gốc, các đường dẫn kế thừa phức tạp và thêm các loại thực thể mới có thể làm cho cơ sở mã phức tạp và không cần thiết. Đồng thời, tính linh hoạt và khả năng sử dụng lại của mã giảm và hiệu suất bị ảnh hưởng. Việc đóng gói dữ liệu trong các kỹ thuật hướng đối tượng cũng là một vấn đề đòi hỏi sự tối ưu để được hiệu suất tốt khi làm việc với một số lượng lớn các thực thể. Để khắc phục những vấn đề này, thiết kế hướng đối tượng truyền thống có thể được thay thế bằng cách tiếp cận thiết kế hướng dữ liệu hơn bằng cách sử dụng mô hình thừa kế thành phần trong đó dữ liệu và logic được tách ra, được gọi là Entity – Component - System (ECS). Trong phương pháp này, các thuộc tính của các thực thể có thể được xác định bằng các thành phần nhỏ, có thể tái sử dụng và không chứa bất kỳ logic nào trong chính chúng. Thay vào đó, logic được xử lý bởi các hệ thống riêng biệt, được khớp với các thành phần và liên tục thực hiện các phương thức bên trong. Một số vấn đề phổ biến hơn gặp phải khi sử dụng các kỹ thuật hướng đối sẽ được đề cập và cách ECS giải quyết các khó khăn đó. Ngoài ra, Em cũng nêu lên các lợi thế đạt được từ việc sử dụng các kỹ thuật thiết kế hướng dữ liệu và cách tiếp cận này có thể được thực hiện ở cấp độ khái niệm.

2.2 Hạn chế của kế thừa OOP

Trong triển khai lập trình hướng đối tượng, mỗi một đối tượng được tạo ra như một thể hiện của một lớp hoặc một tập hợp các thể hiện của các lớp liên kết với nhau. Cách truyền thống để xây dựng một trò chơi là phải có hệ thống phân cấp các đối tượng. Ngay cả những đối tượng đơn giản cũng có thể có một khối lượng lớn các phương thức không được sử dụng, ví dụ:



**Hình 1** Mô hình kế thừa lớp GameObject

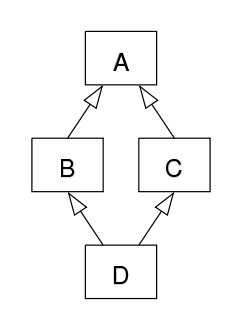
Lớp Animal có 2 lớp con là Bunny, Whale và Killer Whale là lớp con của Whale, với các phương thức khác nhau là hop(), swim(), kill(). Bài toán xuất hiện một đối tượng mới là BunnyKiller gồm 2 phương thức hop() và kill() **vậy nó sẽ kế thừa từ Bunny hay là KillerWhale**

Một số ngôn ngữ chỉ cho phép đơn kế thừa, chỉ có thể kế thừa từ 1 class. Có một cách giải quyết khá đơn giản là khởi tạo 2 phương thức hop() và kill() ở lớp cha Animal, vậy là có thể kế thừa BunnyKiller từ Animal. Tuy nhiên, với các lớp con khác của Animal như Bunny thì lại bị dư thừa đi kill() hay ngược lại Whale lại không sử dụng hop().

Đối với việc đa thừa kế, chúng ta có thể kế thừa BunnyKiller từ Bunny và KillerWhale, nhưng BunnyKiller lại sẽ bị dư phương thức swim(), ngoài ra chúng ta còn phải đối mặt với các vấn đề khác như là:

2.2.1 Mô hình cứng nhắc

Chỉ có KillerWhale hay các lớp con của nó như KillerBunny mới có thể sử dụng phương thức kill(), sẽ rất khó khăn để tạo thêm đối tượng mới để có thể kill(). Phương thức này chỉ có sẵn với các class đã được định nghĩa sẵn mới có thể sử dụng nó. Đối với việc mở rộng số lượng các đối tượng trong dự án, chúng ta sẽ gặp rắc rối vì không biết phải các đối tượng mới ở đâu trong cả hệ thống gia phả của trò chơi.

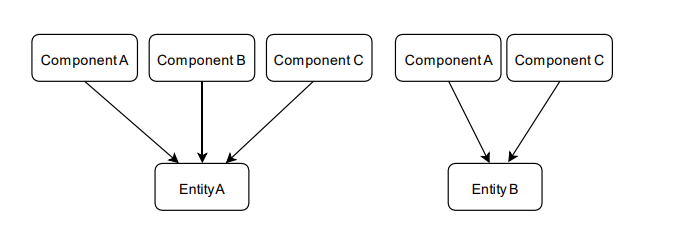


**Hình 2** Kế thừa kim cương

Đây là vấn đề khá phổ biến trong đa kế thừa, class D sẽ là gì? Khi debug lùi sẽ không biết đi ngã rẽ nào là đúng, vì vậy các ngôn ngữ sau này ưu tiên đơn kế thừa và sử dụng interface để giải quyết vấn đề này.

2.2.2 Object-Oriented Composition

Một cách tiếp cận trung gian là tách yếu tố dữ liệu và logic của một đối tượng thành các thành phần tách biệt với thực thể. Trong triển khai này, một thực thể được định nghĩa là một container chứa các component, có thể thêm và xóa trong thời gian chạy chương trình. Nhìn vào hình 1 ta có thể thấy các lớp ké thừa chỉ đơn giản là thêm các tính năng mới cho lớp parent. Việc tách các tính năng này thành các thành phần đã là một giải pháp linh hoạt hơn, vì cách tiếp cận trên không đòi hỏi phải kế thừa cho việc hình thành nên các chức năng cho một đối tượng, giải quyết các vấn đề của kế thừa kim cương. Một hình ảnh của hierarchy:



**Hình 3** Hierarchy của OOC

Trong hình 3, có thể thấy các thực thể bao gồm các thành phần nhỏ, mỗi một component cung cấp cho thực thể một chức năng nhất định, loại bỏ các cấu trúc phân cấp lớp phức tạp, vì các thành phần có thể sử dụng lại giữa các thực thể khác nhau. Mặc dù OOC là một giải pháp phù hợp hơn cho việc xây dựng chương trình có quy mô lớn, phức tạp nhưng vẫn còn chưa đủ, vì nó không hỗ trợ phân tách dữ liệu và logic. Với phương pháp này, tất cả dữ liệu của các đối tượng được tách rời thành các thành phần và có thể được xử lý như các phần riêng biệt giữa các thực thể cùng một lúc thay vì truy cập từng cái một. Điều này cho phép quản lý bộ nhớ tối ưu hóa cao, song song hóa và phân cấp lớp dễ dàng và dễ quản lý.

2.2 ECS

Entity-Component-System (ECS) là một mẫu thiết kế phân phối và thành phần. Nó cho phép tách rời linh hoạt các hành vi theo miền để quản lý các thực thể trong ứng dụng có quy mô lớn. Bằng cách tách dữ liệu khỏi logic, ECS đạt được một hệ thống mô – đun, cho phép lưu trữ dữ liệu với bộ nhớ trong các vùng nhớ liền kề. Với tính mô-đun, các hệ thống có thể tránh được những vấn đề của kế thừa hướng đối tượng truyền thống.

2.2.1 Lịch sử

Một trong những trường hợp sử dụng đầu tiên của thiết kế hướng dữ liệu dựa trên component trong một phần mềm quy mô lớn là một trò chơi năm 1998 có tên là Thief: The Dark Project, nơi mà các nhà phát triển có triết lý tạo ra các thành phần công cụ trò chơi có tính tái sử dụng cao. Cách tiếp cận này hoạt động **rất tốt, đến mức nhóm** có thể sử dụng lại cùng một tệp thực thi thông qua phần lớn sự phát triển cho Thief và System Shock.

Một ví dụ nổi tiếng khác về ECS là một trò chơi năm 2002 có tên Dungeon Siege, nổi bật với một thế giới liền mạch, không có bất kỳ màn hình tải nào, trò chơi này được thực hiện bằng **“DataDriven Game Object System”** một công cụ rất giống với ECS mặc dù thuật ngữ này chưa được đặt ra chính thức. Dungeon Siege nổi bật với hơn 7300 loại đối tượng, cùng với 100.000 đối tượng được đặt trên khắp bản đồ. Một thế giới có quy mô này vào thời điểm đó là một thành tựu đáng chú ý. Tính mô đun của hệ thống cho phép quản lý bộ nhớ linh hoạt trong thời gian chạy

Ngay sau đó, mô hình này đã đượcAdam Martin – một nhà phát triển người Anh đã tạo ra, truyền bá ý tưởng và hướng dẫn của mình về mô hình này. Sau này các nhóm khác phát triển các engine riêng cũng đều dựa trên mẫu Entity-Component-System.

2.2.2 Kiến trúc

Việc triển khai hệ thống thực thể có thể coi là một phần mở rộng của mô hình Object-Oriented Composition, nhưng việc thực hiện và sử dụng nó đòi hỏi một cách suy nghĩ hoàn toàn mới từ người dùng. Trong phương pháp này, các thể hiện của thực thể được giảm từ các đối tượng thành các định danh số nguyên, chỉ phục vụ cho mục đích hợp nhất các thành phần thành một thực thể duy nhất. Các thành phần không chứa xử lý logic mà nó là một tính chất đặc trưng hỗ trợ chuyển đổi dữ liệu đầu vào để thực hiện hành vi. Logic được xử lý bởi các hệ thống riêng biệt, luôn luôn tìm kiếm các component hoạt động.

2.2.3 Entity

Tất cả đối tượng trong game đều là những thực thể (entity), và mỗi thực thể là duy nhất (có một UniqueID). Và những đặc điểm (property), trạng thái (state), phản ứng (behaviour) của các entity được quyết định bởi các Component.

Entity như là một thể hiện được triển khai dưới dạng liên kết duy nhất của các components. Entity sẽ không có dữ liệu hay hành vi thực tế nào, nó chỉ là nơi liên kết các tính chất của component để hỗ trợ một hành vi nào đó.

Một số triển khai của ECS cho phép sửa đổi các component của mỗi entity trong lúc runtime.

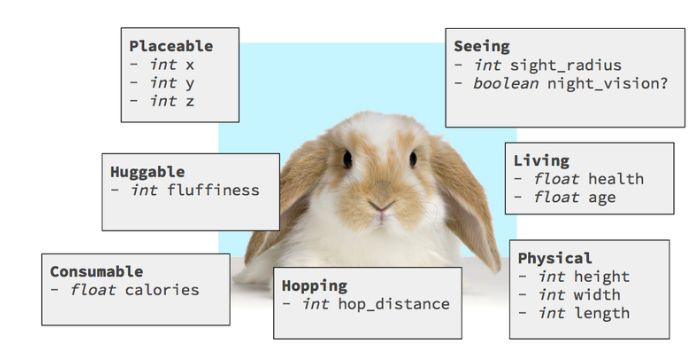
Bird (con chim trong Flappy Bird) là một entity, nó được “tổng hợp” bởi: thành phần hiển thị (Renderable), có khả năng di chuyển (Movable), có yếu tố vật lý (Collier, Body)

Pipe (ống vật cản) cũng là entity, nó bao gồm: Thành phần hiển thị (Renderable), yếu tố vật lý (Collider, Body), và không có yếu tố về di chuyển (Movable) và AI.

2.2.4 Component

Components là các đoạn dữ liệu của ECS. Chúng là các phần nhỏ, có tính chất riêng và có thể tái sử dụng. Chúng được sử dụng để xác định các thuộc tính của thực thể và tương tác với các thực thể khác.

Chúng chứa tất cả dữ liệu, có thể thêm vào các thực thể để hỗ trợ thực hiện các hành vi. Một component được thêm vào đối tượng nó sẽ thể hiện một tính chất duy nhất, một component tự nó sẽ không có hành vi. Thông thường nó được triển khai như một struct hoặc dictionary.



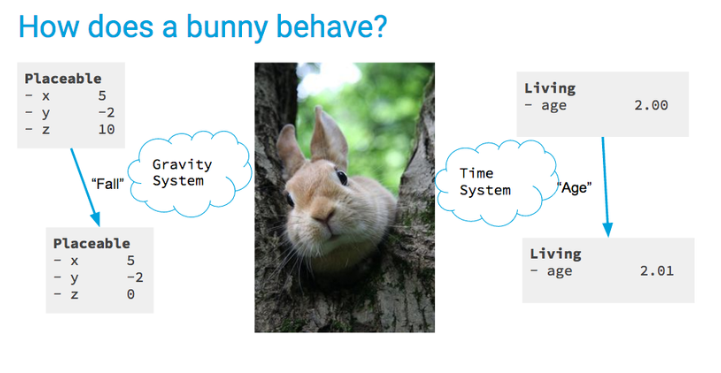
**Hình 4** Các component cụ thể

Theo hình 4, chúng ta có thể biểu diễn đối tương Bunny với các component thể hiện tích chất của nó như Hopping, Seeing, Living.

Không có thành phần nào trong số này chứa bất kỳ logic nào trong chính chúng.

2.2.5 System

System liệt kê các components và cập nhật trạng thái của chúng theo quy tắc nội bộ hoặc tác động từ các sự kiện bên ngoài để thể hiện một hành vi hay thay đổi từ trạng thái này sang trạng thái khác.



**Hình 5** Systems của một entity

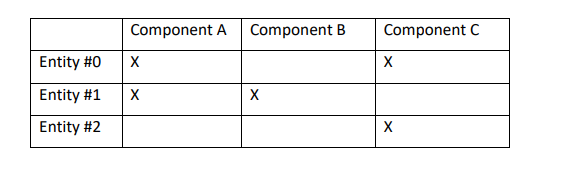
**Gravity System:**thay đổi các trạng thái của Placeable component, thể hiện hành vi đang rơi xuống của bunny

**Time System:**cập nhật thay đổi của Living component, thể hiện sự sống của bunny

**=>** Có các System dành riêng cho các hành vi cụ thể, system hoạt động trên các Components chứ không phải các Entity.

2.3 Composing the model.

Lập trình một hệ thống với Entity-Component-System giống như lập trình hướng dữ liệu. Một thể hiện của thực thể hoạt động tương tự một khóa trong cơ sở dữ liệu, giống như bất kỳ hệ thống quản lý cơ sở dữ liệu quan hệ nào. Từ quan điểm trừu tượng, truy cập vào một thành phần của một thực thể cụ thể như một truy vấn cơ sở dữ liệu. Khái niệm này được hiểu rõ hơn ở bảng sau:

**Hình 6** Bảng quan hệ Entity-Component.

Có thể thấy các thành phần khác nhau được gắn với các thực thể và cách thể hiện của một thành phần được chia sẻ giữa nhiều thực thể.

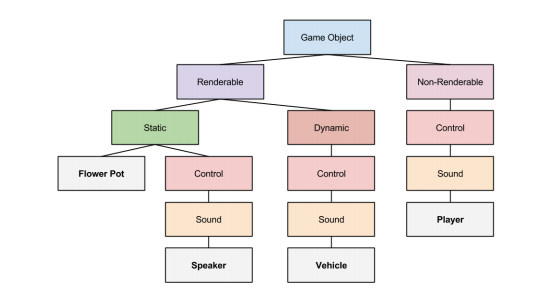
2.4 Thiết kế hướng dữ liệu

Khi phát triển phần mềm lớn, người ta thường gặp vấn đề hiệu năng khiến thời gian thực hiện chương trình bị chậm, bị giật cục trong thời gian dài. Có rất nhiều lý do, nhưng rõ rang nhất là thuật toán thực thi không được tối ưu hóa, nhưng điều này không phải lúc nào cũng đúng. Ngoài ra còn lý do khá phổ biến nữa là quản lý bộ nhớ không hiệu quả. Để có thể quản lý dữ liệu hiệu quả hơn, có thể thực hiện chuyển đổi từ tư duy hướng đối tượng sang thiết kế hướng dữ liệu (DOD). DOD nhằm mục đích chuyển hoàn toàn trọng tâm từ các đối tượng sang xử lý dữ liệu thực tế. Thay vì để một đối tượng cô lập sẽ làm mọi việc, thay vào đó, hãy để các phương thức thực hiện các phép biến đổi theo các cách chung và cố gắng sắp xếp dữ liệu một cách hiệu quả cho phần cứng nhất có thể.

Ở quy mô lớn hơn, việc dữ liệu được định vị tối ưu trong bộ nhớ có thể có các tác động mạnh đến hiệu suất của phần mềm, do đó điều quan trọng là ta cần chú ý tới các loại dữ liệu và cách xử lý dữ liệu.

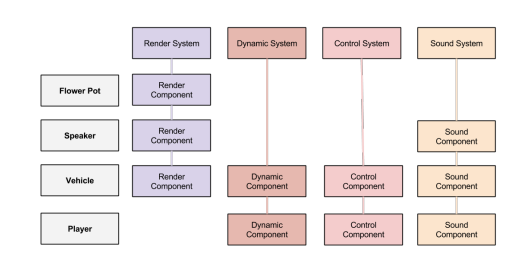
Bố cục lý tưởng có thể đạt được bằng cách chia các đối tượng thành các thành phần khác nhau và nhóm các thành phần đó lại với nhau trong bộ nhớ phân theo loại. Điều này dẫn đến dữ liệu đồng nhất và xử lý tuần tự. Cách tiệp cận này hoạt động rất tốt trên các nhóm đối tượng lớn, không giống như OOP, chủ yếu tập trung vào một đối tượng duy nhất tại thời điểm đó. Vì lý do này, ECS rất hiệu quả và thường được sử dụng trong phát triển trò chơi. Do tính chất mô-đun của ECS, mô hình cho phép phân cấp thực thể cực kỳ linh hoạt và giảm bớt kết nối mạng và tuần tự hóa dữ liệu.

2.5 Phân tích hệ thống ECS



**Hình 7** Hệ thống phân cấp OOP

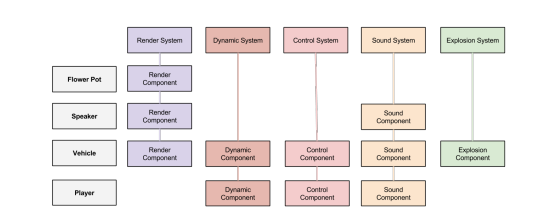
Hình 7 cho thấy một cây phân cấp tiềm năng với các đối tượng trò chơi được thêm vào. Càng kết hợp nhiều tính năng thêm vào, các đối tượng trò chơi, cấu trúc phân cấp sẽ càng phức tạp. Điều này khiến mã khó quản lý và tổ chức. Một vấn đề chính nữa của hệ thống phân cấp này là xử lý các đối tượng trò chơi vì sẽ rất khó để tạo ra một chức năng duy nhất chấp nhận các đối tượng trò chơi khác nhau có chức năng cụ thể. Các giải pháp như nạp chồng hàm rất khó cho việc bảo trì, mở rộng vì mọi loại đối tượng sẽ chứa một chức năng cụ thể. Với hệ thống này khiến chúng ta rất khó để mở rộng hệ thống do mỗi nhánh con lại phụ thuộc rất nhiều vào cấu trúc của nhánh cha của nó. Thêm một tính năng mới hay thay đổi một tính năng sẽ gặp khó khăn hoặc không thể mở rộng được. Điều này khiến cho việc xây dựng engine gặp khó khắn, buộc chúng ta phải xây dựng lại hệ thống theo quan điểm mới.



**Hình 8** Hệ thống chuyển đổi về ECS

Hình 8 thể hiện thiết kế phân cấp trước đó được chuyển đổi thành ECS. Đây là một ví dụ cho việc tách biệt các tính năng và có tính tổ chức mạnh mẽ. Thay vì đi theo một cây phân cấp phức tạp, một bảng đơn giản được dùng để xem tổng quan những tính năng mà một đối tượng trò chơi có.

Khó khăn của hệ thống phân cấp OOP là bảo trì và mở rộng, nhưng ECS đã khắc phục được nhược điểm này



**Hình 9** Hệ thống ECS thêm chức năng

Một lợi ích bổ sung của hệ thống ECS là tính module và khả năng mở rộng của hệ thống. Vì hệ thống và các thành phần tương đối tách biệt và độc lập nên nó dễ dàng cho phép thêm các tính năng mới. Ví dụ, thêm âm thanh cũng như thêm một thành phần của âm thanh. Thêm một tính năng mới như vụ nổ thì ta sẽ tạo ra một Explosion System và Explosion Component. Như hình 6, việc tạo thêm các thành phần mới có tác động rất ít vào cấu trúc của engine đã tạo ra.

Mặc dù giải quyết được các vấn đề mà OOP gặp phải, nhưng ECS cũng có những vấn đề của riêng mình. Tách biệt và độc lập có cả điểm mạnh lẫn điểm yếu trong hệ thống ECS. Điểm yếu ở đây xảy ra khi thiết kế yêu cầu chia sẻ các thành phần và giao tiếp chéo. Vấn đề với các thành phần được chia sẻ là quy trình, nếu chia sẻ các thành phần với nhau, nó sẽ đi ngược lại với giá trị cốt lõi của ECS là sự tách biệt và độc lập. Ngoài ra, thứ tự của các thực thể xử lý cũng đóng một vai trò quan trong. Ví dụ như một engine cùng với chuyển động, va chạm và render, thứ tự mà các thực thể này xử lý sẽ thay đổi kết quả mà chúng ta có thể nhìn thấy. Ví dụ như một vật chuyển động, render và va chạm có thể sinh ra trường hợp bị mắc vào một vật khác khi xảy ra va chạm.

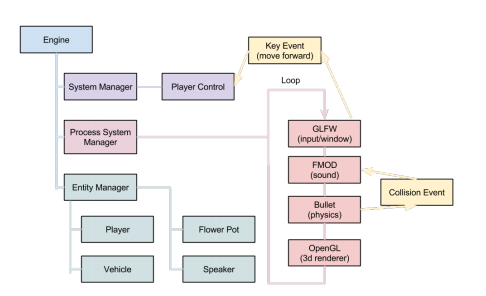
Sự giao tiếp hệ thống chéo là cần thiết vì mỗi một thực thể chứa rất nhiều thành phần và hệ thống. Ví dụ, khi một quả bóng va chạm sẽ gây ra tiếng động. Vấn đề ở đây là sự va chạm và hệ thống âm thanh là tách biệt và độc lập với nhau. Vậy làm sao ta biết được lúc nào va chạm để gửi tới thông điệp cho hệ thống âm thanh xử lý. Dù bằng cách nào đi nữa, việc thiết kế một phương pháp để kết nối các hệ thống trong khi giữ cho chúng độc lập và tách biệt là rất khó khăn.

2.6 Cupcake ECS

Kiến trúc Cupcake ECS được thiết kế để trở thành một module trong hệ thống ECS với mục đích dễ dàng thêm hay xóa các hệ thống mà không ảnh hưởng tới kiến trúc ban đầu. Điều này yêu cầu hệ thống phải hoàn toàn độc lập, tách biệt và không được sử dụng tham chiếu tới các hệ thống khác. Ý tưởng này sẽ tạo ra một engine mà người dùng có thể sử dụng các tính năng mới và tái sử dụng lại nó.

ECS Framework đã chưa các tiện ích để quản lý hệ thống, các thực thể và các thành phần, và một khung có sẵn để tao ra các hệ thống và thành phần mới. Tiếp theo người dùng có thể chọn một danh sách các plugin (các hệ thống) cho nhũng yêu cầu về chức năng. Ví dụ, người dùng có thể chọn một danh sách như FMOD cho âm thanh, OpenGL cho 3D rendering, và Bullet cho việc xử lý vật lý. Và cuốicùng, một tập lệnh cài đặt sẽ biên dịch các plugin thành một thư viện tĩnh và tập hợp các tệp tiêu đề, điều đó sẽ tạo ra một công cụ có tính tùy biến cao.

2.6.1 Architecture



**Hình 10** Kiến trúc Cupcake

Kiến trúc cupcake được chia thành nhiều phần, engine, manager và systems. Hình 10 cho thấy kiến ​​trúc cơ bản của Cupcake. Engine này là giao diện lớp trên cùng cho tất cả các system, manager và entity. Nó chịu trách nhiệm chạy vòng lặp trò chơi quản lý các hệ thống và thực thể. Engine quản lý các thực thể bằng cách cung cấp khả năng thêm và xóa các thực thể trong khi duy trì một id duy nhất cho mỗi thực thể. Các hệ thống được xử lý bởi engine mỗi vòng lặp trò chơi. Các hệ thống chịu trách nhiệm quản lý các thành phần và thực hiện một số phương pháp trừu tượng. Các phương thức trừu tượng bao gồm init, release và update (cho các hệ thống xử lý). Một ví dụ về một hệ thống FMOD.

Hàm init sẽ khởi tạo tệp thư viện FMOD

Hàm release sẽ giải phóng tất cả các tài nguyên gắn liền với hệ thống.

Hàm update chịu trách nhiệm cập nhật vị trí âm thanh cho các đối tượng chuyển động.

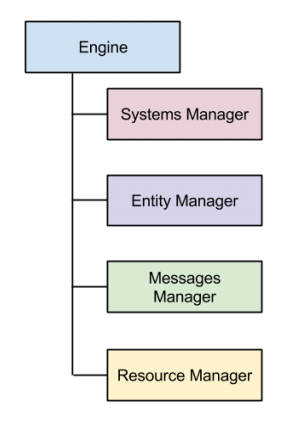
2.6.2 Cupcake Solutions

Cupcake có cùng các vấn đề chung với bất kỳ hệ thống ECS ​​nào trong các component được chia sẻ và giao tiếp hệ thống chéo. Để giải quyết vấn đề chia sẻ component, Cupcake có một bộ component bên ngoài bên ngoài engine. Các thành phần bên ngoài này được chuyển đến System khi được khởi tạo. Điều này cho phép nhiều hệ thống có cùng một bộ component mà không phải yêu cầu bất kỳ phương pháp nào để giữ dữ liệu đồng bộ. Ngoài ra, nó duy trì sự độc lập của các hệ thống với nhau. Một ví dụ về chia sẻ một thành phần là rendering. Bằng cách tham chiếu cùng một danh sách các component, mọi thay đổi được thực hiện theo vị trí của vật lý sẽ tự động cập nhật cho việc renderer. Giao tiếp chéo của các hệ thống được giải quyết với một message system bên ngoài. Hệ thống này bao gồm handlers, triggers và message. Triggers là một phần logic được thực thi mỗi vòng lặp cho các điều kiện cụ thể. Dựa trên các điều kiện, một tin nhắn sẽ được gửi đến hệ thống tin nhắn. Xử lý tin nhắn là một đối tượng được tạo ra cho các hệ thống để thực hiện logic dựa trên các thông điệp. Một ví dụ về một người chơi di chuyển đòi hỏi ba phần. Một kích hoạt để thực hiện dựa trên đầu vào của người chơi. Khi kích hoạt khởi tạo đầu vào của người chơi, nó sẽ gửi thông báo di chuyển thích hợp vào hệ thống, chẳng hạn như di chuyển về phía trước. Tin nhắn chuyển tiếp sẽ được một trình xử lý gửi đến hệ thống vật lý.

2.7 ARTEMIS ECS FRAMEWORK

Artemis ECS Framework được tạo bởi Gamadu được thiết kế để phát triển trò chơi. Thiết kế tương đối giống như Cupcake. Các thực thể là các id duy nhất, các thành phần là dữ liệu và hệ thống xử lý logic bằng cách sử dụng dữ liệu. Artemis Framework ban đầu được viết bằng Java, tuy nhiên phân tích này sẽ đi qua cổng Vinova Mobile C ++ của Artemis Framework. Từ cổng Java sang C ++, một số tính năng không được triển khai nên phân tích này chỉ chính xác với cổng C ++.

2.7.1 Architecture



**Hình 11** Kiến trúc của ECS Artemis Framework

Artemis ECS Framework chứa một World Object là một giao diện của framework. World Object này chứa các trình quản lý và vòng lặp trò chơi. Các trình quản lý được sử dụng trong frame này dành cho systems, entities, groups và tags. Các trình quản lý và thực thể tương tự như Cupcake trong đó chúng quản lý các hệ thống và thực thể tồn tại. Group và Tag quản lý là các hệ thống đơn giản được sử dụng để sắp xếp các thực thể lại với nhau. Điều này được thực hiện bằng cách liên kết một số chuỗi với một nhóm các thực thể hoặc một thực thể duy nhất. Entity Manager có trong Artemis là duy nhất vì nó lưu trữ tất cả các thành phần có trong engine. Điều này được thực hiện bằng mảng hai chiều, trong đó có định danh đầu tiên là component type và định danh thứ hai là entity id. Entity Manager cũng quản lý các id duy nhất cho các thực thể đang hoạt động.

Các hệ thống trong Artemis quản lý một danh sách các thực thể cần xử lý. Điều này được thực hiện bằng cách chỉ định một định danh cho từng component type. Sử dụng component type, các thực thể có thể được hệ thống lọc ra để chỉ các thực thể hợp lệ được xử lý. Để truy cập dữ liệu, “caller” phải tạo các trình ánh xạ để có thể lấy được các thành phần từ Entity Manager.

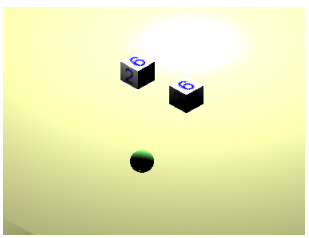
2.7.2 Analysis

Artemis có một giải pháp duy nhất để sửa các lỗi chia sẻ các component. Bằng việc đính kèm các component tới Entity Manager, điều này cho phép tất cả hệ thống có quyền trùy cập vào các component. Điều này hữu ích khi “caller” cần xóa một thực thể vì trình quản lý thực thể có thể tìm kiếm thông qua danh sách các component và xóa tất cả dữ liệu liên quan đến thực thể.

Một vấn đề với Artemis là sự thiếu giao tiếp hệ thống. Điều duy nhất được chia sẻ giữa các hệ thống là các component có thể được sử dụng làm thông điệp. Tuy nhiên, đây có thể là một phương thức không hiệu quả vì việc tạo và xóa các thành phần rất tốn kém.

Cuối cùng, giới hạn lớn nhất của Artemis là giới hạn type component và system. Các id liên quan đến các thành phần và hệ thống được thực hiện với bit. Các bit được giới hạn trong khoảng nhất định, điều này tạo ra một vấn đề trong đó quá trình lọc bị lỗi khi người dùng vượt quá giới hạn component hoặc system. Các component mới được gán loại không xác định có thể cho phép các thực thể không hợp lệ được chuyển đến các hệ thống.

2.7.3 Các dự án sử dụng ECS



**Hình 12** Cup cake test app



**Hình 13** Carrota Bunny

**Carrota Game**

Sử dụng Artemis Framework, ECS Engine được phát triển cho dự án. Với engine này, một trò chơi phòng thủ FPS đã được tạo ra. Các tính năng trong công cụ này bao gồm: 2D, 3D texture rendering, sound, collision, physics, animation. Mỗi tính năng tương đối độc lập với nhau.

2.8 Lợi ích của ECS

Trong lập trình hướng đối tượng, việc tách một quá trình có thể xuất hiện lỗi đồng bộ gây ra bởi các luồng khác nhau do cố truy cập đồng thời dữ liệu. Vì dữ liệu được xử lý theo nhóm, nên dễ dàng phân chia dữ liệu giữa các luồng khác nhau và sẽ không bị lỗi đồng bộ dữ liệu. Một điểm mạnh khác của thiết kế hướng dữ liệu là khả năng sử dụng bộ đệm đươc tối ưu hóa. Trong phần cứng để đạt được hiệu năng tốt là sắp xếp dữ liệu trong bộ nhớ có thể sử dụng nhiều lần. Nếu dữ liệu được đặt liền kề trong bộ nhớ, sẽ được xử lý với bộ đệm nên hiệu suất sẽ được tăng cao.

Ưu điểm của ECS bao gồm:

Đảm bảo nguyên tắc thực hiện một nhiệm vụ duy nhất.

Khả năng kết hợp hay định nghĩa đối tượng trong lúc runtime.

Dễ dàng testing, với mỗi component hay system.

Tính song song, có thể xây dựng nhiều System làm việc cùng một lúc.

Phân tách dữ liệu và hành vi, có thể mở rộng các hành vi khác nhau trên cùng một dữ liệu.

2.9 Giới thiệu về SDL

**Simple DirectMedia Layer (SDL):**là một thư viện đa nền tảng, bao gồm các API để thao tác với âm thanh, bàn phím, chuột, joystick, graphics hardware thông qua OpenGL và Direct3D.

Thư viện SDL được viết bằng C chuẩn, nhưng hỗ trợ tốt với C++ và nhiều ngôn ngữ lập trình khác như:C#,Java, Pascal, PHP, Python,...

SDL hỗ trợ trên các hệ điều hành như: Windows, Mac OS X, Linux, iOS và Android.

Các game sử dụng thư viện SDL như: Syberia II, 7 Grands Step, Secret Maryo Chronicles. Dưới đây là một số hình ảnh về game được tạo bởi thư viện SDL.



**Hình 14** Secret Maryo Chronicles



**Hình 15** 7 [Unknown Horizons](https://vi.wikipedia.org/w/index.php?title=Unknown_Horizons&action=edit&redlink=1)



**Hình 16** [Trine](https://vi.wikipedia.org/w/index.php?title=Trine_(Computerspiel)&action=edit&redlink=1)

2.10 Lịch sử và kiến trúc

2.10.1 Lịch sử

[Sam Lantinga](https://vi.wikipedia.org/w/index.php?title=Sam_Lantinga&action=edit&redlink=1) là người tạo ra thư viện này vào năm [1998](https://vi.wikipedia.org/wiki/1998), lúc ông còn làm cho hãng Loki Software. Ông nảy ra ý định này khi mang các chương trình Windows vào môi trường Macintosh. Sau đó ông dùng thư viện này để đem chương trình giải trí [Doom](https://vi.wikipedia.org/wiki/Doom) sang hệ điều hành BeOS (Xem [mã nguồn của chương trình Doom](https://vi.wikipedia.org/w/index.php?title=M%C3%A3_ngu%E1%BB%93n_c%E1%BB%A7a_ch%C6%B0%C6%A1ng_tr%C3%ACnh_Doom&action=edit&redlink=1)). Một vài thư viện tự do khác như [SMPEG](https://vi.wikipedia.org/w/index.php?title=SMPEG&action=edit&redlink=1) và [OpenAL](https://vi.wikipedia.org/w/index.php?title=OpenAL&action=edit&redlink=1) có thể hoạt động chung với SDL.

SDL được chia thành nhiều hệ thống con (subsystem) như: Video, Audio, CD-ROM, Joystick và Timer. Bên cạnh những hệ thống con cơ bản này, do đây là thư viện cấp thấp, còn có một số thư viện chính thức riêng biệt, cung cấp các chức năng khác. Bao gồm:

SDL\_image: cung cấp các hàm để đọc các định dạng ảnh phổ biến ngày nay như: [JPEG](https://vi.wikipedia.org/wiki/JPEG), [PNG](https://vi.wikipedia.org/wiki/PNG), [BMP](https://vi.wikipedia.org/wiki/BMP).

SDL\_mixer: các hàm audio dùng để hòa âm (mixing) hay đọc các tập tin âm thanh hay nhạc như: [WAV](https://vi.wikipedia.org/w/index.php?title=WAV&action=edit&redlink=1), [OGG](https://vi.wikipedia.org/w/index.php?title=OGG&action=edit&redlink=1), [MP3](https://vi.wikipedia.org/wiki/MP3)

SDL\_net: hỗ trợ lập trình mạng đa hệ điều hành

SDL\_ttf: hỗ trợ hiển thị các phông chữ TTF (true type font)

SDL\_rtf: hỗ trợ hiển thị định dạng RTF đơn giản

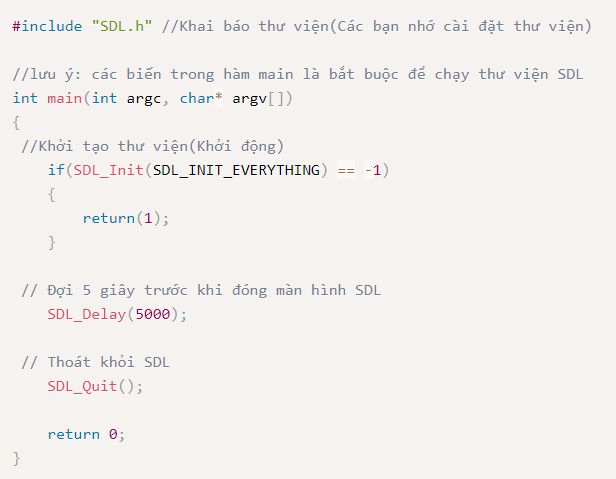
2.10.2 Kiến trúc

SDL thật sự là một gói bao bọc (wrapper) xung quanh các chức năng xác định của [hệ điều hành](https://vi.wikipedia.org/wiki/H%E1%BB%87_%C4%91i%E1%BB%81u_h%C3%A0nh). Mục đích chính của SDL là lập thành một giao diện lập trình cốt lõi đa hệ điều hành phổ biến cho việc truy cập chức năng đa phương tiện.

Do cách thiết kế của SDL, hầu hết mã nguồn của thư viện được tách thành nhiều phần cho từng hệ điều hành. Khi SDL được biên dịch cho một hệ điều hành, các phần thích hợp được chọn cho hệ điều hành đó.

Trong sơ đồ, đối với hệ điều hành Windows, SDL thật sự bao bọc xung quanh [DirectX](https://vi.wikipedia.org/wiki/DirectX). Phiên bản cũ của SDL sử dụng DirectX 5. Phiên bản ổn định hiện tại của SDL sử dụng DirectX 7. Trong môi trường đồ họa [X11](https://vi.wikipedia.org/w/index.php?title=X11&action=edit&redlink=1), bao gồm Linux, SDL sử dụng Xlib để giao tiếp với hệ thống X11 khi làm việc với đồ họa và sự kiện.

Dưới đây là một chương trình rất đơn giản dùng thư viện SDL



**Hình 17** Chương trình khởi động hệ thống đa phương tiện

2.11 Các khái niệm cơ bản trong SDL

2.11.1 Tile

Tile là khái niệm chỉ đến một ảnh có kích thước nhỏ (thường là ảnh bitmap), thường là hình vuông hoặc hình chữ nhật. Độ dài các cạnh của tile (tính theo pixel) thường là một ước số của khung nhìn (cửa sổ màn hình). Các hình ảnh lớn trong game được tạo nên từ các tile nhỏ. Việc chia nhỏ các hình ảnh lớn này thành các tile nhỏ hơn sẽ thuận lợi cho việc lưu trữ và linh hoạt trong việc tạo ra các kiểu hình ảnh đồ họa trong game, ngoài ra còn tiết kiệm bộ nhớ rất nhiều và tốc độ chương trình cũng được cải thiện nhờ tốc độ load ảnh.



**Hình 18** Sprite

Với những ưu điểm như vậy, tile thường được ứng dụng trong việc tạo map (bản đồ) cho game 2D.

2.11.2 Sprite sheet

Sprite, khác với tile, là một hình ảnh toàn vẹn của một đối tượng. Mỗi đối tượng động có một tập các sprite riêng tương ứng với từng dạng chuyển động, mỗi sprite là một hình ảnh mô tả một trạng thái trong dạng chuyển động đó. Việc thay đổi liên tiếp các sprite này với một khoảng thời gian chờ hợp lý giữa các sprite sẽ tạo ra các hình ảnh chuyển động với cảm giác thật. Các hình ảnh chuyển động thường thấy trong game, đặc biệt là game 2D như nhân vật di chuyển, nước chảy, hay các vụ nổ do va chạm, đều có thể được thể hiện bằng cách dùng sprite.



**Hình 19** Sprite sheet

2.11.3 Frame

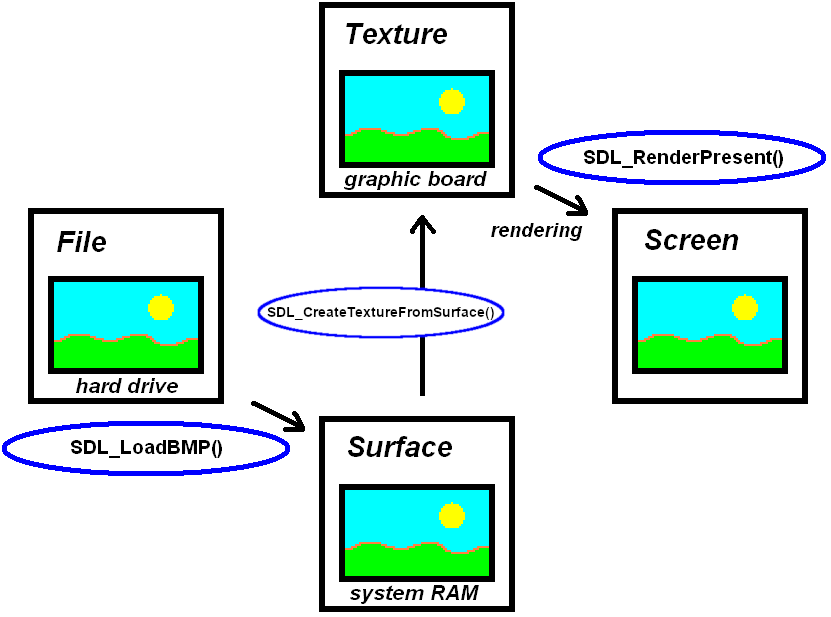
Tất cả các thiết bị hiển thị hình ảnh chuyển động hiện nay như TV, monitor… đều dựa trên khả năng lưu ảnh của mắt người. Sau khi hình ảnh biến mất mắt của chúng ta còn lưu lại tại vỏng mạc 1/24 giây. Vì vậy để tạo các đoạn film người ta chỉ cần thay đổi liên tục các hình ảnh theo tốc độ 24 hình/giây hoặc cao hơn. Trong game cũng áp dụng kỹ thuật này để tạo chuyển động và mỗi lần cập nhật màn hình gọi là 1 frame

2.11.4 Map

Map là vùng thể hiện tất cả các đối tượng trong game, một phần hoặc toàn bộ bề mặt của map được hiển thị trên màn hình. Map chứa tất cả các đối tượng khác. Tất cả các đối tượng chỉ có thể di chuyển trong phạm vi của map và có mức độ ưu tiên khi xuất hiện trên map khác nhau Cách tổ chức map trong game là phần rất quan trọng, đối với mỗi game khác nhau lại có các tổ chức khác nhau.

2.11.5 Surface

Surface là khái niệm chỉ một vùng bộ nhớ để lưu dữ liệu hình ảnh. Hình ảnh được lưu là một vùng hình chữ nhật, có thể là hình ảnh được load lên từ một file hình ảnh nào đó hoặc là sự tổng hợp của nhiều hình ảnh khác nhau. Khi cần các surface có thể được vẽ lên màn hình để hiển thị hình ảnh đồ họa trong game.



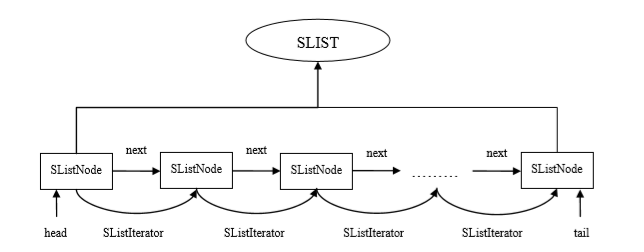
**Hình 20** Quy trình loading ảnh

2.11.6 Structure Containers

Đây là kĩ thuật dùng để tích hợp các modules và đồng bộ hóa các tất cả các đối tượng trong chương trình. Kĩ thuật này sử dụng một template danh sách liên kết đơn có kiểu dữ liệu của lớp cha để lưu trữ thông tin của các thuộc tính và duyệt qua tất cả các phương thức ảo (Virtual method) của các lớp con kế thừa lớp cha nhằm thực hiện tính đa hình

Các thành phần của constructures container:

* SlistNode
* SList
* SListIterator



**Hình 21** Template danh sách liên kết

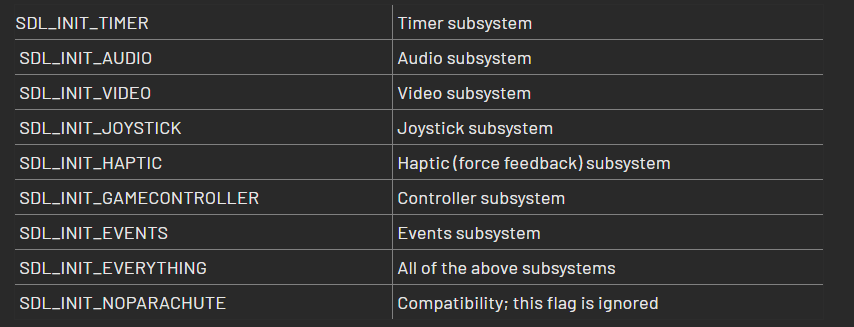
2.12 Các vấn đề gặp phải trong quá trình sử dụng SDL

2.12.1 Khởi tạo cửa sổ

Khởi tạo thư viện SDL có cung cấp cho chúng ta một hàm để khởi tạo thư viện. Hàm này có prototype như sau:

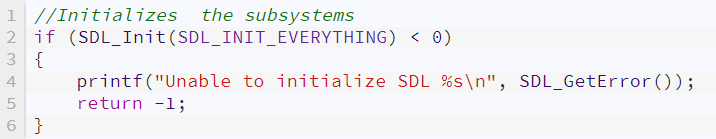
Hàm trả về 0 nếu khởi tạo thành công và ngược lại trả về -1 nếu khởi tạo thất bại. Có thể sử dụng hàm SDL\_GetError(); để lấy về thông báo lỗi nếu khởi tạo thất bại.

Tham số của SDL\_init là một tham số flags kiểu int, nằm trong những giá trị sau đây hoặc có thể dùng toán tử ‘|’ để kết hợp nhiều lựa chọn.



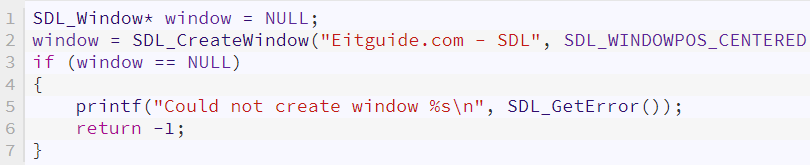
**Hình 22** Tham số của SDL\_INIT

Bược đầu tiên, ta sẽ khởi tạo thư viện SDL



**Hình 23** Khởi tạo thư viện SDL

Sau khi đã khởi tạo ta cần một đối tượng để quản lý cửa số là SDL\_Window được khởi tạo như sau:



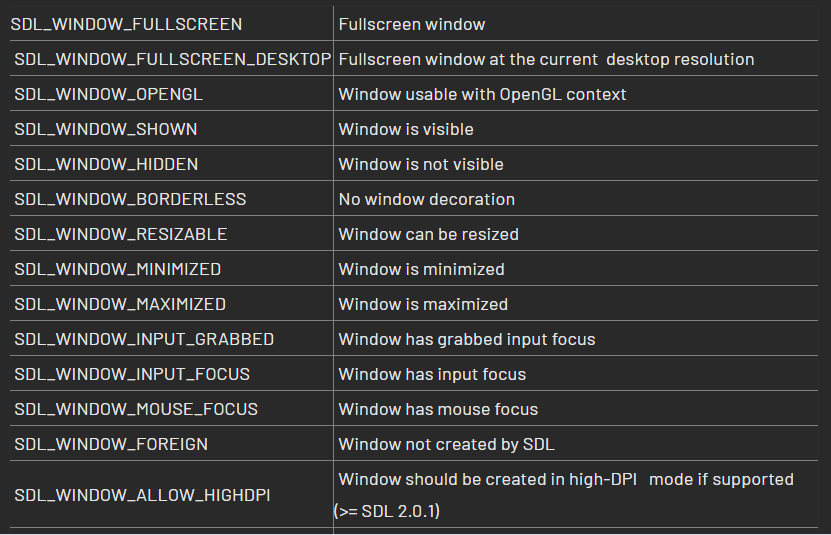
**Hình 24** Khởi tạo Window

Hàm SDL\_CreateWindow trả về NULL nếu thất bại, hàm có prototype như sau:



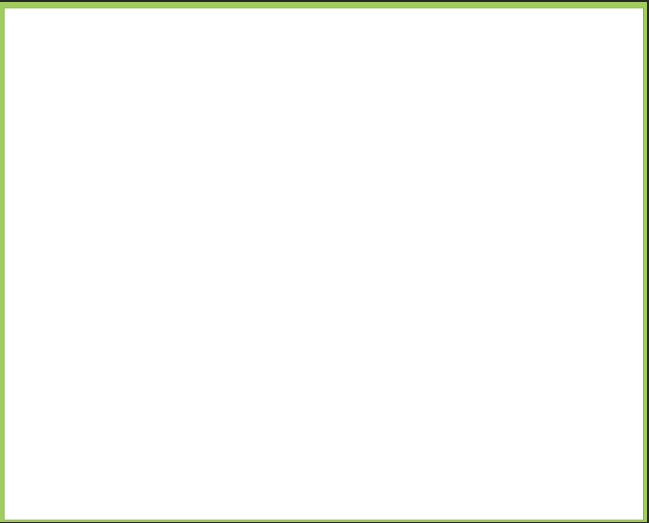
**Hình 25** Prototye của SDL\_CreateWindow

Tham số cuối cùng có thể dùng toán tử ‘|’ để kết hợp nhiều lựa chọn dưới đây:



**Hình 26** Tham số Unit32

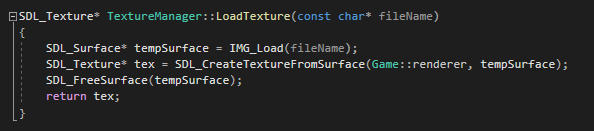
Kết quả sau khi biên dịch và chạy chương trình:



**Hình 27** Cửa sổ window

2.12.2 LoadTexture

Mỗi đối tượng trò chơi đều cần texture để có thể hiển thị trong game. Ta cần xây dựng một hàm có thể tạo ra một surface image bao phủ màn hình windows với đối số đầu vào là tên của ảnh background. Và hàm này sẽ thực hiện việc đọc file ảnh và trả kết quả kiểu dữ liệu SDL\_Texture.



**Hình 28** Load Texture trong SDL

Các đối tượng sử dụng trong mã như hình 16 bao gồm:

SDL\_Window: là struct giữ tất cả các thông tin của cửa sổ mà chúng ta tạo ra như size, postion, border, fullscreen.

SDL\_Renderer: là struct xử lý tất cả các công việc rendering lên cửa sổ SDL\_Window.

SDL\_Surface: chứa một tập hợp các pixel (a collection of pixels) để có thể render lên cửa sổ sử dụng software rendering (CPU).

SDL\_Texture: chứa một tập hợp các pixel (a collection of pixels) để có thể render lên cửa sổ sử dụng hardware rendering (GPU).

2.12.3 Kỹ thuật Color Key

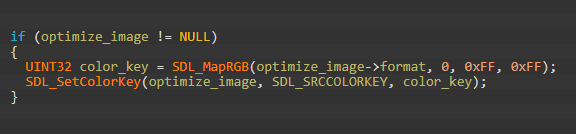
Sau khi thực hiện load ảnh như trên, màn hình sẽ hiển thị hình ảnh như hình dưới đây.

Tuy nhiên chúng ta sẽ gặp vấn đề là background của hình ảnh này màu xanh. Chúng ta cần phải xóa background xanh này đi.



**Hình 29** Load ảnh nhân vật lỗi

Kỹ thuật color key giúp chúng ta xóa background. Thêm đoạn code dưới này ngày sau khi load texture



**Hình 30** Mã xóa background ảnh bị lỗi

Đoạn code này sẽ thực hiện transparent background của nhân vật.

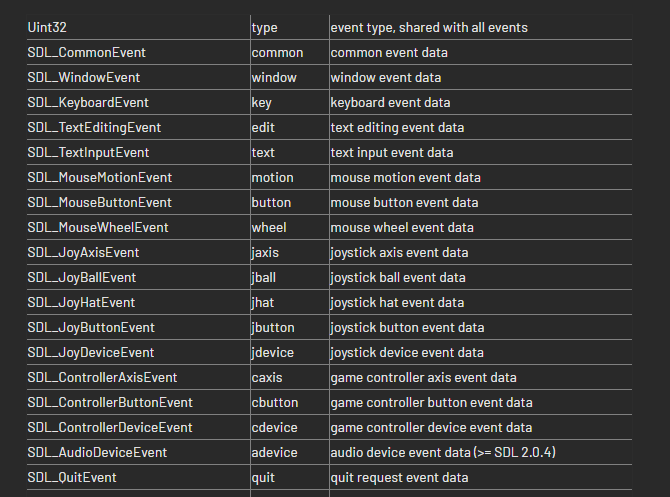
Với điều kiện mã màu truyền vào phải trùng với mã màu của background. Sau khi sử dụng các mã trên ta thu được kết quả như hình dưới đây.



**Hình 31** Background nhân vật đã được xóa.

2.12.4 Sự kiện trong SDL

SDL có cung cấp một union có tên là SDL\_Event để chứa tất cả các thông tin của tất cả các event trong bộ thư viện này. Cấu trúc này như sau:



**Hình 32** Các event trong SDL

Tất cả các event của Windows đều được lưu ở hàng đợi queue. SDL có định nghĩa cho chúng ta một hàm để lấy ra event từ hàng đợi đó. Hàm này có prototype như sau:



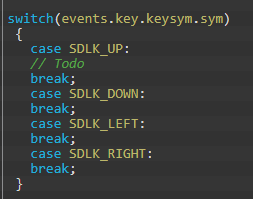
**Hình 33** Prototype của SDL\_PollEvent

Có tham số truyền vào là một con trỏ đối tượng SDL\_Event. Trả về 1 nếu có event tồn tại trong queue và ngược lại trả về 0 nếu queue rỗng. Và các chức năng là lấy thông tin của event ở queue và đổ dữ liệu vào đối tượng SDL\_Event sau đó xóa event đó khỏi queue.

2.12.5 Bắt sự kiện nhấn hoặc thả ra.

Bắt sự kiện sử dụng **events.key.keysym.sym.**

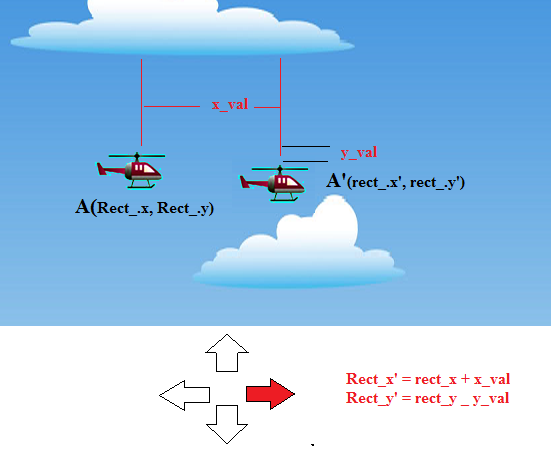
**Ví dụ ta muốn bắt sự kiện: Left-Right-Up-Down để di chuyển nhân vật.**



**Hình 34** Xử lý bắt sự kiện trong SDL

2.12.6 Xử lý di chuyển cho đối tượng nhân vật

Nguyên lý cơ bản là sử dụng Left, Right, Up, Down để di chuyển nhân vật ra bốn hướng. Khi một phím được bấm, thì x\_val và y\_val trong lớp đối tượng sẽ được thay đổ, tăng lên hoặc giảm đi.



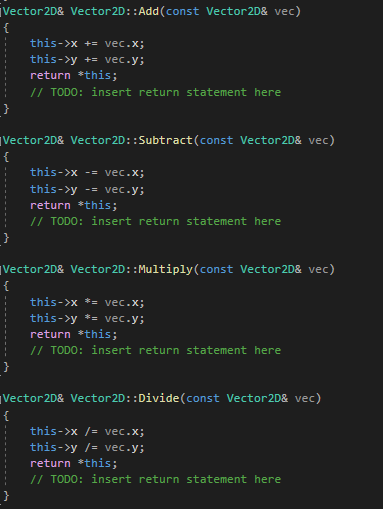
**Hình 35** Xử lý di chuyển của đối tượng.

Hàm move sẽ thực hiện nhiệm vụ thay đổi vị trí hiện tại của đối tượng từ A đến A’. Tọa độ từ A đến A’ sẽ bằng tọa độ từ A cộng với x\_val và y\_val. Vì thế mà vật đi nhanh hay đi chậm là phụ thuộc vào x\_val và y\_val. Thường thì game sẽ có một biến speed thay đổi dựa trên độ khó để tăng thêm trải nghiệm của người chơi. Nhân vật sẽ được giới hạn di chuyển trong phạm vi màn hình, nếu x\_val và y\_val lớn vượt qua giới hạn thì phải xử lý chặn.

2.12.7 Xây dựng lớp Vector2D

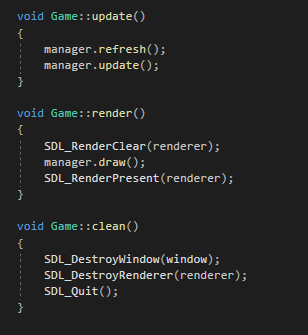
Nguyên lý cơ bản: Lớp Vector2D này chứa hai giá trị x và y là thể hiện của đối tượng trong không gian hai chiều.

Lớp Vector2D này giúp khởi tạo, xác định vị trí của đối tượng và di chuyển tới các đối tượng được trỏ đến. Ngoài ra có thêm các hàm tăng giảm tùy ý như Add, Subtract, Divide, Multiply.



**Hình 36** Vector2D Class

2.12.8 Update, Render, Clean

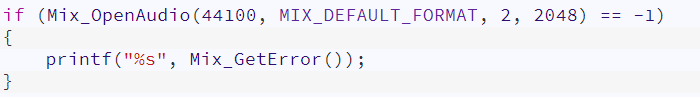


**Hình 37** Các hàm cập nhật trong game

Nguyên lý cơ bản là có một Manager quản lý tất cả các thực thể trong game, hàm update() sẽ làm mới và cập nhật trạng thái hiện tại của các thực thể qua mỗi frame. Sau đó hàm render() sẽ xóa trạng thái cũ của các thực thể, sau khi xóa xong, đối tượng manager sẽ thực hiện hàm draw() để cập nhật lại hình ảnh mới cho đối tượng rồi sau đó render ra để hiển thị trên windows. Hàm clean() là làm sạch bộ nhớ sau khi thoát game.

2.12.9 Play Audio trong SDL

Tích hợp SDL\_mixer vào project, trước khi sử dụng function mà SDL\_mixer cung cấp, ta cần khởi tạo thư viện này:



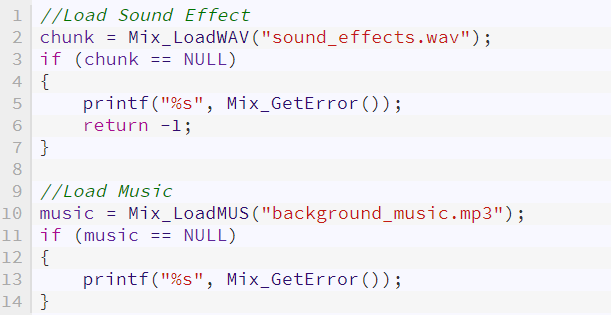
**Hình 38** Khởi tạo thư viện SDL\_mixer

SDL\_mixer có hỗ trợ chúng ta hai cấu trúc để quản lý Music và Sound Effect là Mix\_music và Mix\_Chunk.



**Hình 39** Khai báo biến quản lý sound và music

Với cấu trúc trên thì ta có các hàm load tương ứng là Mix\_LoadWAV và Mix\_LoadMUS



**Hình 40** Load sound và music

Với music và sound effect thì SDL\_mixer có cung cấp các hàm bổ trợ play, pause, resume, stop cũng như các hàm kiểm tra trạng thái.

Với Music, các hàm bổ trợ play music:

int Mix\_PlayMusic(Mix\_Music \*music, int loops): Phát music được load bởi hàm Mix\_LoadMUS, vói tham số truyền vào là con trỏ tới đối tượng Mix\_Music, và số lần lặp, nếu truyền vào -1 sẽ lặp vô tận.

void Mix\_PauseMusic(): Tạm dừng phát music.

void Mix\_ResumeMusic(): Tiếp tục phát lại music sau khi pause.

int Mix\_HaltMusic(): Dừng phát music.

Và các hàm kiểu tra trạng thái:

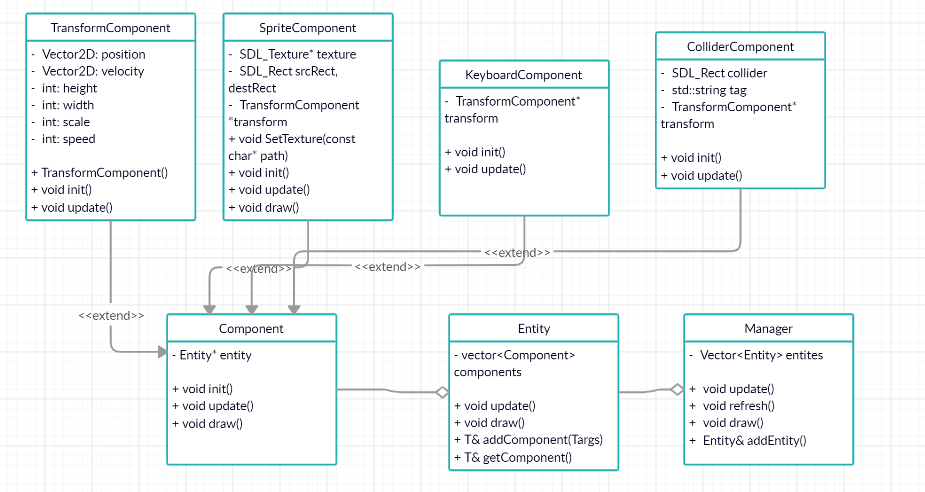
int Mix\_PlayingMusic(): Trả về 1 nếu music đang phát, ngược lại trả về 0.

int Mix\_PausedMusic(): Trả về 1 nếu music tạm dừng, ngược lại trả về 0.

Chương 3 Xây dưng engine và phát triển game Flappy Bird

3.1 Xây dựng engine game sử dụng mã nguồn mở SDL2.

3.1.1 Các thành phần cấu tạo có trong engine game.

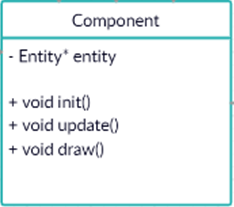


**Hình 41** Mô hình ECS cơ bản cấu thành nên engine

Mỗi một engine được tạo nên đều có những thành phần cơ bản với các nghiệp vụ tương ứng và cần thiết cho mỗi đối tượng trò chơi, mà cụ thể ở đây là Transform Component, Sprite Component, Keyboard Component, Collider Component. Nhưng để có thể chia thành các module xử lý riêng ta cần một nền tảng cơ sở để khởi tạo và mở rộng thêm.

3.1.2 Phát triển engine

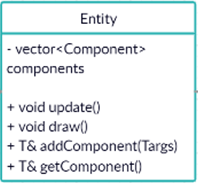
3.1.2.1 Component



**Hình 42** Class Component

Thiết lập một Base Component trong engine game. Gồm các hàm cơ bản như init(), update() hay draw(). Mỗi một component đều có một entity, thực thể mà component đó được thêm vào.

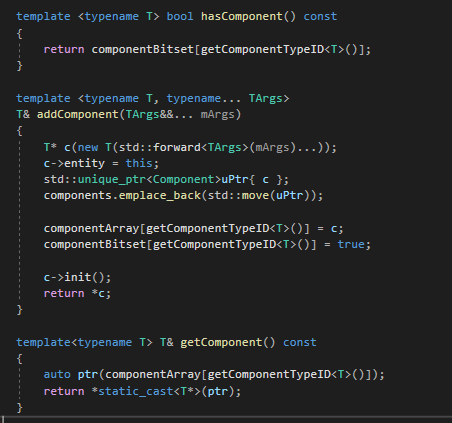
3.1.2.2 Entity



**Hình 43** Class Entity

Entity là thực thể ở trong trò chơi, mỗi một entity sẽ chứa các component tùy vào yêu cầu của mỗi thực thể mà sẽ có các component tương ứng.

Như trên hình, hàm update() hay draw của entity sẽ gọi đến tất cả các component mà entity được gắn để có thể cập nhật trạng thái liên tục.

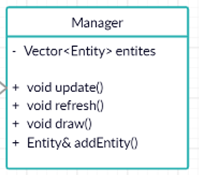


**Hình 44** Các thành phần trong Entity

Hàm AddComponent: đây là hàm quan trong nhất, giúp cho chúng ta có thể thêm các thành phần tùy ý vào một thực thể. Ví dụ: Flappy Bird có chuyển động ta sẽ gắn cho nó TransformComponent, chim có thể va chạm với các vật cản, nên ta cần gắn cho nó một CollisionComponent để detect được va chạm …

Hàm GetComponent(): là hàm trả về thành phần theo tham số T truyền vào. Ví dụ: trong một trò chơi bắn nhau, khi bị bắn trúng ta cần trừ máu của thực thể bị bắn, ta cần phải lấy được Health Component để trừ máu.

3.1.2.3 Manager



**Hình 45** Class Manager

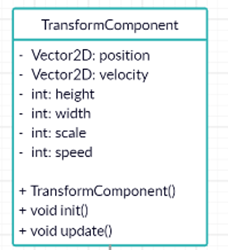
Manager đóng một vai trò quan trọng trong game. Nó như là đầu não của game, chứa tất cả các thực thể có trong game. Và việc các thực thể hoạt động ra sao, được cấu hình như nào thì đều thông qua class Manager.

Các hàm update(), draw(), refresh() được cập nhật liên tục theo frame để hiển thị trạng thái hiện tại của các thực thể.

Hàm addEntity(): Mỗi một entity được sinh ra sẽ được thêm vào và chịu sự quản lý của Manager.

Ba class cơ bản đã được hình thành để làm cơ sở phát triển ra các chức năng của game bao gồm:

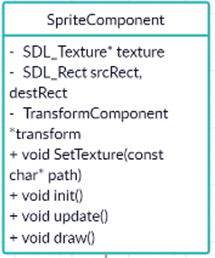
3.1.2.4 TransformComponent



**Hình 46** Class Transform Component

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

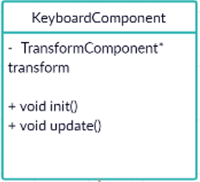
3.1.2.5 SpriteComponent



**Hình 47** Class SpriteComponent

Class SpriteComponent này cho phép các thực thể load các hình ảnh 2D có thể là ảnh đầy đủ hay một bộ phận.

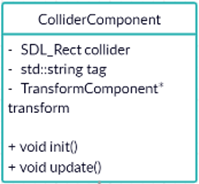
3.1.2.6 KeyboardComponent



**Hình 48** Class KeyboardComponent

KeyboardComponent rất quan trọng, vì khi người chơi thao tác, tất cả các action, event keyboard đều phải lấy được thông tin action, event để update dữ liệu cho người chơi. Ví dụ như nhân vật sang trái, sang phải thì keyboard phải lấy được thông tin lúc người chơi bấm vào button\_left và button\_right.

3.1.2.7 ColliderComponent



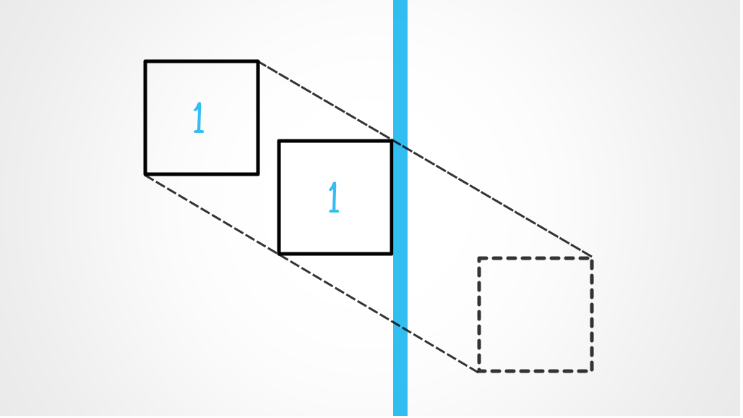
**Hình 49** Class ColliderComponent

Va chạm và xử lý va chạm trong game là một thành phần không thể thiếu khi lập trình game. Va chạm trong game xảy ra khi chúng ta có 2 đối tượng đi vào không gian của nhau. Ví dụ: trúng đạn, trúng mũi tên, chạm vạch đích, đều là các sự kiện va chạm và khi lập trình game chúng ta phải xử lí các va chạm đó.

ColliderComponent được xây dựng lên để giải quyết vấn đề đó. Việc khám phá ra va chạm đòi hỏi việc cập nhật trạng thái của đối tượng trong từng frame. SDL\_Rect đóng một vai trò như giới hạn các biên của thực thể, bên trong nó là khoảng không gian chứa thực thể, và nếu có một thực thể khác lấn vào không gian đó thì va chạm sẽ được hình thành.

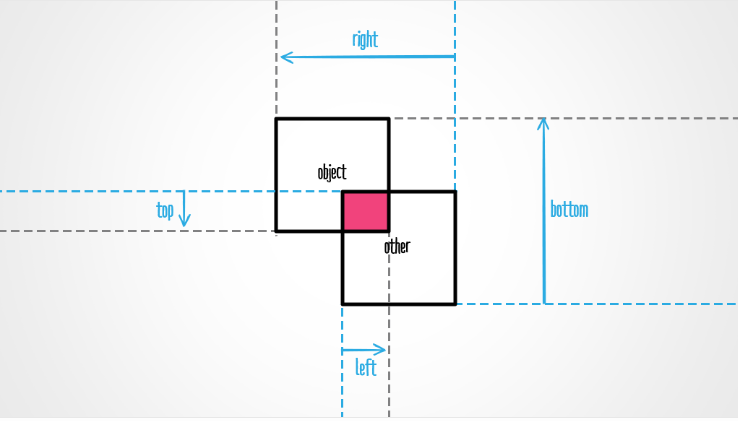
Kỹ thuật phát hiện va chạm dựa vào hộp bao AABB.

AABB là viết tắt của Axis-Aligned Bounding Box, nó là thuật toán xét va chạm giữa các cạnh của **hình chữ nhật** mà ở đây các cạnh này nó **song song** với cùng **hệ trục tọa độ**. Vậy cơ bản là xét xem 2 hình chữ nhật nó có chồng lên nhau hay không. Chúng ta sẽ xem trước ở frame kế tiếp hình chữ nhật mình có va chạm hay không. Có thì sẽ xử lý nó theo ý muốn của game.



**Hình 50** Mô phỏng va chạm

Ví dụ: Ở **frame đầu tiên**, hình chữ nhật chưa có va chạm, **frame tiếp theo** hình chữ nhật di chuyển 1 đoạn và va chạm với thanh màu xanh. Ta sẽ dự đoán trước và quyết định vị trí kế tiếp của nó, như ở đây nó sẽ đi đoạn ngắn hơn thay vì như bình thường sẽ là ở vị trí nét đứt.



**Hình 51** Mô phỏng va chạm giữa Object và Other

**Xét va chạm AABB**

Để xem 2 hình chữ nhật có va chạm hay không thì chúng ta sẽ xét cặp cạnh của 2 hình trên như sau:

2 hình đè lên nhau thì mình luôn có **ĐỒNG THỜI**:

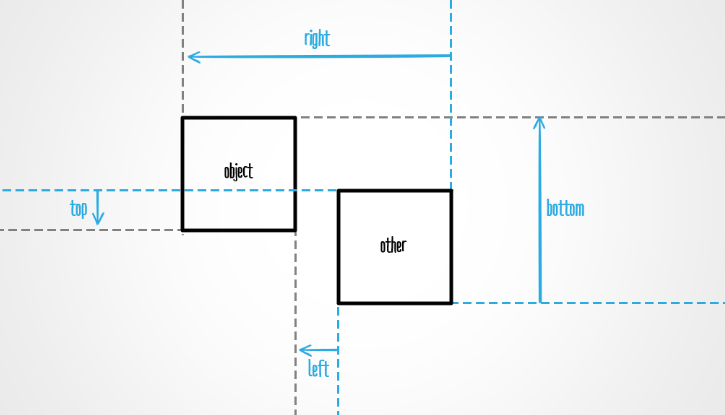
Other.left <= object.right

Other.right >= object.left

Other.top >= object.bottom

Other.bottom <= object.top

Như vậy nếu **1 trong 4**điều kiện trên **sai** thì 2 hình **không có va chạm** với nhau.

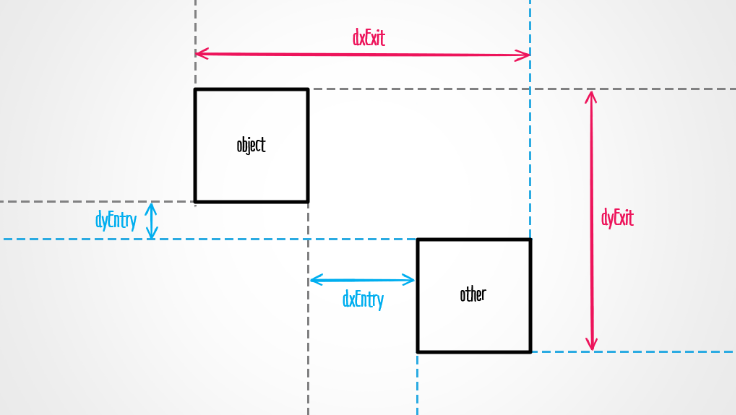


**Hình 52** Mô phỏng không va chạm giữa Object và Other

Ví dụ ở đây ta có **left** của **other** lớn hơn **right** của **object**, nên không có va chạm.

**Xét va chạm với Swept AABB**

Đầu tiên, ta tìm khoảng cách giữa các cạnh của hai hình.



**Hình 53** Hiển thị các thông số cho Swept AABB

Trong đó,

**dxEntry**, **dyEntry**: là khoảng cách cần đi để các bắt đầu va chạm.

**dxExit**, **dyExit**: là khoảng cách cần đi kể từ lúc này để khi hết va chạm.

Cũng như top, left, right, bottom của AABB cơ bản phía trên. Ta xét thêm hướng của vận tốc, để nó đồng bộ với dấu lúc sau tính thời gian không ngược khi có va chạm. Từ khoảng cách và vận tốc, ta tìm thời gian để bắt đầu và kết thúc va chạm. Để xảy ra va chạm, cả hai trục x và y phải đồng thời xảy ra va chạm, vậy ta lấy thời gian bắt để đầu va chạm lớn nhất.

Còn khi hết va chạm, chỉ cần 1 trong 2 trục thoát khỏi là được, nên ta lấy thời gian kết thúc va chạm nhỏ nhất giữa 2 trục x, y.

Có được thời gian rồi thì ta bắt đầu xét va chạm:

Lớn hơn 1.0f: frame tiếp theo nó vẫn chưa thể va chạm.

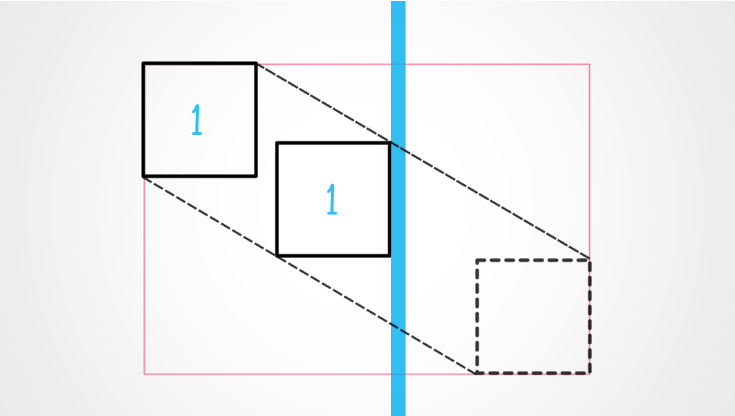
Thời gian để kết thúc va chạm nhỏ hơn 0.0f: 2 hình chữ nhật đang đi ra xa nhau.

Thời gian để kết thúc va chạm phải lớn hơn thời gian để va chạm (va chạm xong rồi thì sau đó mới hết chứ đúng không).

Khi có thể va chạm ta sẽ trả về thời gian va chạm đó, còn không trả về 1.0f.

**Xử lý va chạm**

Dùng broad-phasing xét xem có có thể va chạm trong frame tiếp theo không.



**Hình 54** Dự đoán va chạm của đối tượng với vật cản

Ta tạo 1 hình chữ nhật dựa trên **vị trí ban đầu** và **kế tiếp**, sau đó lấy hình chữ nhật đó xét xem có chồng lên với hình kia không. Nếu có thì va chạm, còn không thì chắc chắn không thể nên không cần xét tiếp.

3.2 Phát triển game bằng engine tự xây dựng và Unity Engine

3.2.1 Giới thiệu về game Flapy Bird

**Flappy Bird là một trò chơi điện tử trên điện thoại do Nguyễn Hà Đông, một lập trình** viên ở Hà Nội phát triển. Trò chơi được trình bày theo phong cách side-scroller, trong đó người chơi điều khiển một chú chim, cố gắng vượt qua các hàng ống màu xanh lá cây mà không chạm vào chúng.

3.2.2 Các đối tượng trong game.

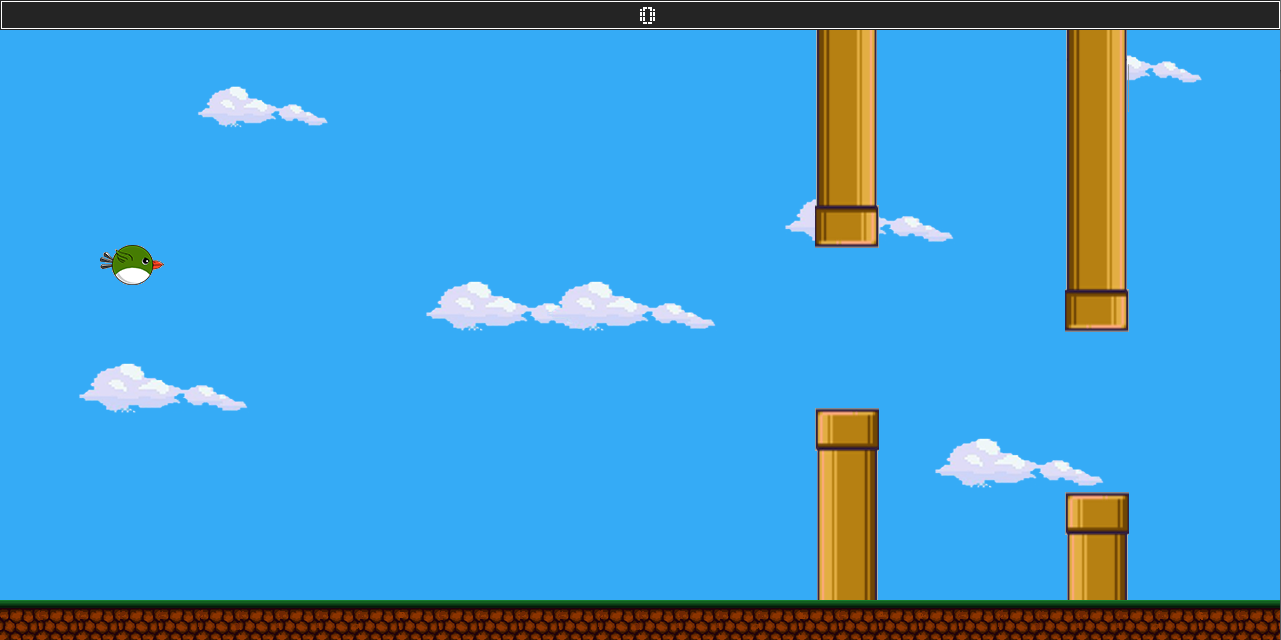
**Giao diện Menu Game**

Màn hình menu có giao diện là một chú chim và các ống nâu, hai đối tượng chính của game, ngoài ra có hai option là chơi hoặc thoát game.



**Hình 55** Menu Game

**Giao diện game play**



**Hình 56** Giao diện GamePlay

Giao diện game play thể hiện vị trí của chú chim, các ống vật cản, nền đất và điểm của người chơi.

Giao diện GameOver



**Hình 57** Giao diện Game Over

Hiển thị cho người chơi biết mình đã chơi thua để có thể chơi lại.

**Đối tượng Bird**



**Hình 58** Đối tượng Bird

Đối tượng chim sẽ được người chơi điều khiển trực tiếp để có thể đi qua được các ống vật cản.

**Đối tượng ống vật cản**



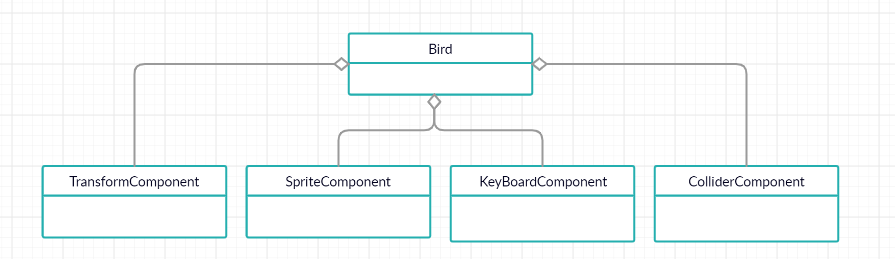
**Hình 59** Giao diện GamePlay

3.2.3 Phát triển game bằng engine tự xây dựng.

Các thực thể chính trong game:

**Bird:** với các thể hiện gồm có hình ảnh lúc bay và lúc bị rơi xuống đất, có di chuyển, có va chạm và có event để người dùng thao tác.

Từ đó ta dễ dàng có thể xây dựng được thực thể Bird như hình sau



**Hình 60** Các thành phần cấu tạo nên Bird

Thực thể Bird sẽ được thêm 4 thành phần gồm:

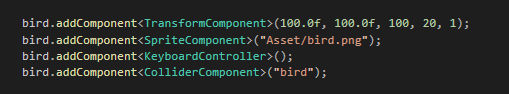
**TransformComponent**: xác định vị trí của Bird.

**SpriteComponent**: Load hình ảnh của Bird khi đang bay hay lúc chết.

**KeyboardComponent**: Bắt các event mà người dùng thao tác.

**ColliderComponent**: Khám phá ra va chạm với các vật cản

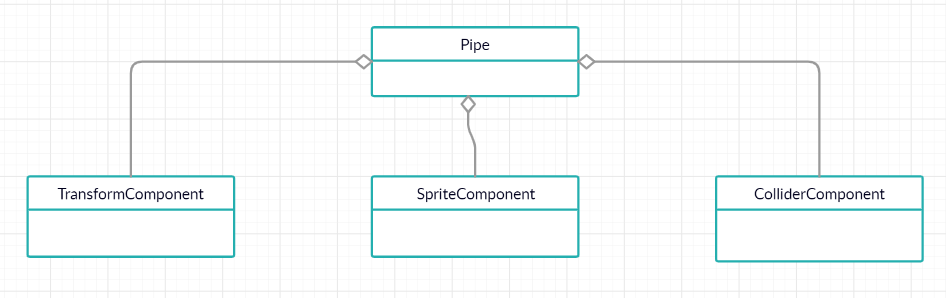
Bird sẽ được thêm các component như sau:

  
**Hình 61** Cách thêm các thành phần vào Bird

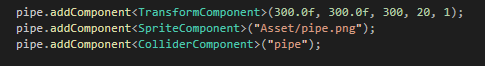
Khi thêm các component, ta cũng đồng thời thêm đầu vào cho Bird như vị trí, load sprite, thêm keyboard để người dùng thao tác, và bird tag để có thể xử lí va chạm.

**Pipe:** Khác với Bird, Pipe không cần đến thao tác của người dùng.

Xây dựng thực thể Pipe:

 **Hình 62** Các thành phần cấu tạo nên Pipe

Pipe sẽ được thêm các component như sau:

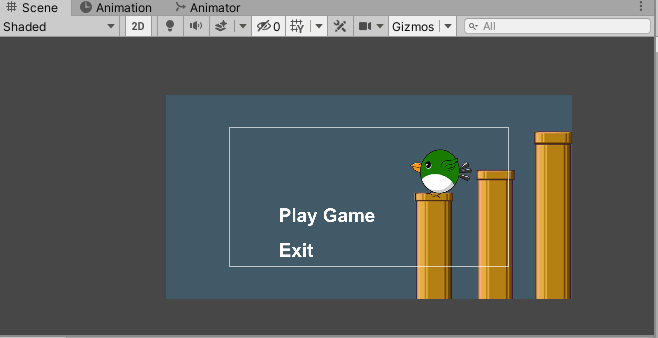
 **Hình 63** Cách thêm các thành phần vào Bird

3.2.4 Phát triển game bằng unity engine

Unity là một “cross- flatform game engine” tạm hiểu là công cụ phát triển game đa nền tảng được phát triển bởi Unity Technologies. Game engine này được sử dụng để phát trển game trên PC, consoles, thiết bị di động và trên websites.

3.2.4.1 Các thành phần có trong Unity Editor

**Cửa số Scene**

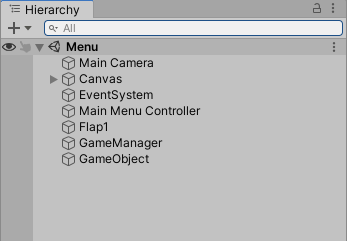


**Hình** 64 Cửa sổ scene trong Unity

Phần này phần hiển thị các đối tượng trong scenes một cách trực quan, có thể lựa chọn các đối tượng, kéo thả, phóng to, thu nhỏ, xoay các đối tượng ...

Phần này có để thiết lập một số thông số như hiển thị ánh sáng, âm anh, cách nhìn 2D hay 3D ... -Khung nhìn Scene là nơi bố trí các Game Object như cây cối, cảnh quan, enemy, player, camera, … trong game. Sự bố trí hoạt cảnh là một trong những chức năng quan trọng nhất của Unity.

**Cửa sổ Hierachy**



**Hình** 65 Cửa sổ Hierachy

Tab hierarchy là nơi hiển thị các Game Object trong Sences hiện hành. Khi các đối tượng được thêm hoặc xóa trong Sences, tương ứng với các đối tượng đó trong cửa sổ Hierarchy.

Tương tự trong tab Project, Hierarchy cũng có một thanh tìm kiếm giúp quản lý và thao tác với các Game Object hiệu quả hơn đặc biệt là với các dự án lớn.

**Cửa sổ Game**

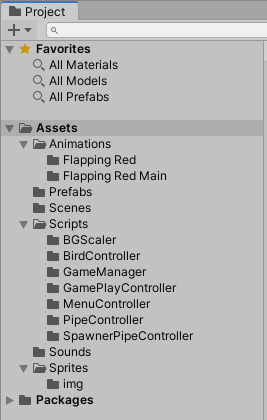


**Hình 66** Cửa sổ Game

Đây là mạn hình demo Game, là góc nhìn từ camera trong game.

Thanh công cụ trong cửa sổ game cung cấp các tùy chỉnh về độ phân giải man hình, thông số (stats), gizmos, tùy chọn bật tắt các component...

**Cửa sổ Project**



**Hình 67** Cửa sổ project

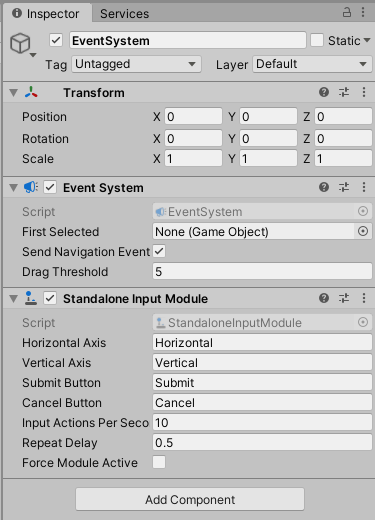
Đây là cưa sổ explorer của Unity, hiển thị thông tin của tất cả các tài nguyên (Assets) trong game của bạn.

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

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

Đường dẫn của thư mục tài nguyên hiện tại. Chúng ta có thể dễ dàng tiếp cận các thư mục con hoặc thư mục gốc bằng cách click chuột vào mũi tên hoặc tên thư mục.

**Cửa sổ Inspector**



**Hình 68** Cửa sổ Inspector

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

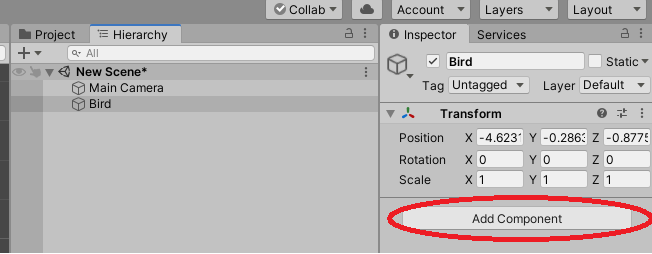
Mọi thuộc tính thể hiện trong Inspector đều có thể dễ dàng tuỳ chỉnh trực tiếp mà không cần thông qua một kịch bản định trước. Tuy nhiên Scripting API cung cấp một số lượng nhiều và đầy đủ hơn do giao diện Inspector là có giới hạn.

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

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

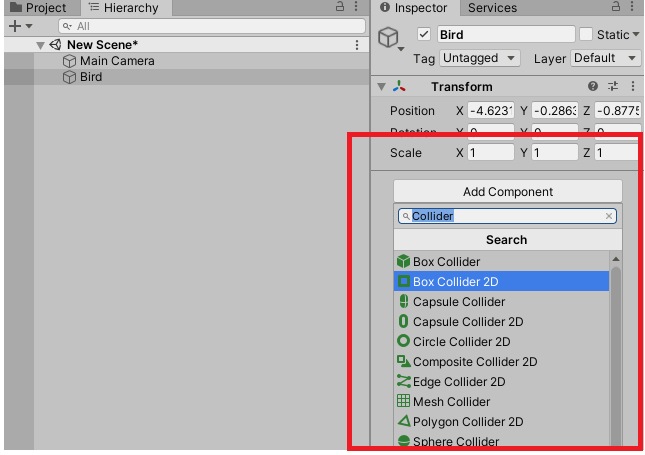
3.2.4.2 Xây dựng game FlappyBird

**Bird:** Khởi tạo thực thể Bird gồm có 4 component là Transform Component, ColliderComponent, SpriteComponent và KeyBoardComponent.



**Hình 69** Thêm thành phần cho Bird

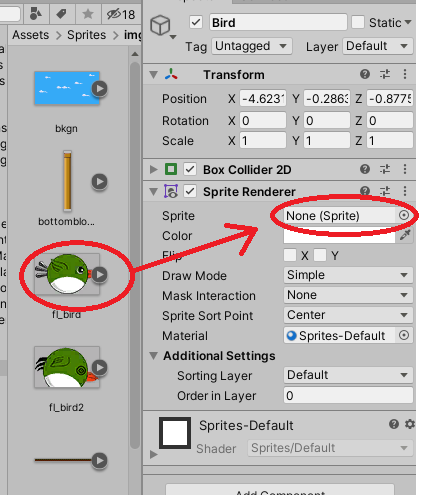
Khởi tạo một Game Object Bird, khi được sinh ra, Bird sẽ được mặc định có Transform Component. Do Unity đã cũng cấp sẵn các thư viện về Collider hay SpriteRender, nên ta chỉ việc AddComponent cho Bird.



**Hình 70** Thêm collider cho Bird

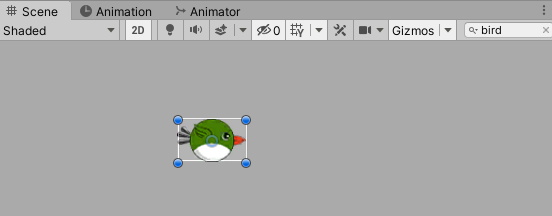
Unity cung cấp rất nhiều các thành phần xử lý va chạm cho cả 2D lẫn 3D. Ở đây chúng ta chọn BoxCollider2D.

Sau khi chọn xong, chúng ta tiếp tục thêm SpriteComponent( trong trường hợp ở Unity nó được gọi là SpriteRenderer). Nếu muốn thêm ảnh cho Bird, ta có thể thực hiện kéo thả ngay trên Editor như hình dưới đây



**Hình 71** Load ảnh cho Sprite Renderer

Dựa trên Editor mà khi phát triển game trên Unity rất là dễ dàng, vì ta có thể kéo thả và xem trực tiếp được trên Scene mà không cần phải run project.



**Hình 72** Hình ảnh đã được load trên scene

Chương 4 Kết quả đạt được

4.1 Kết quả

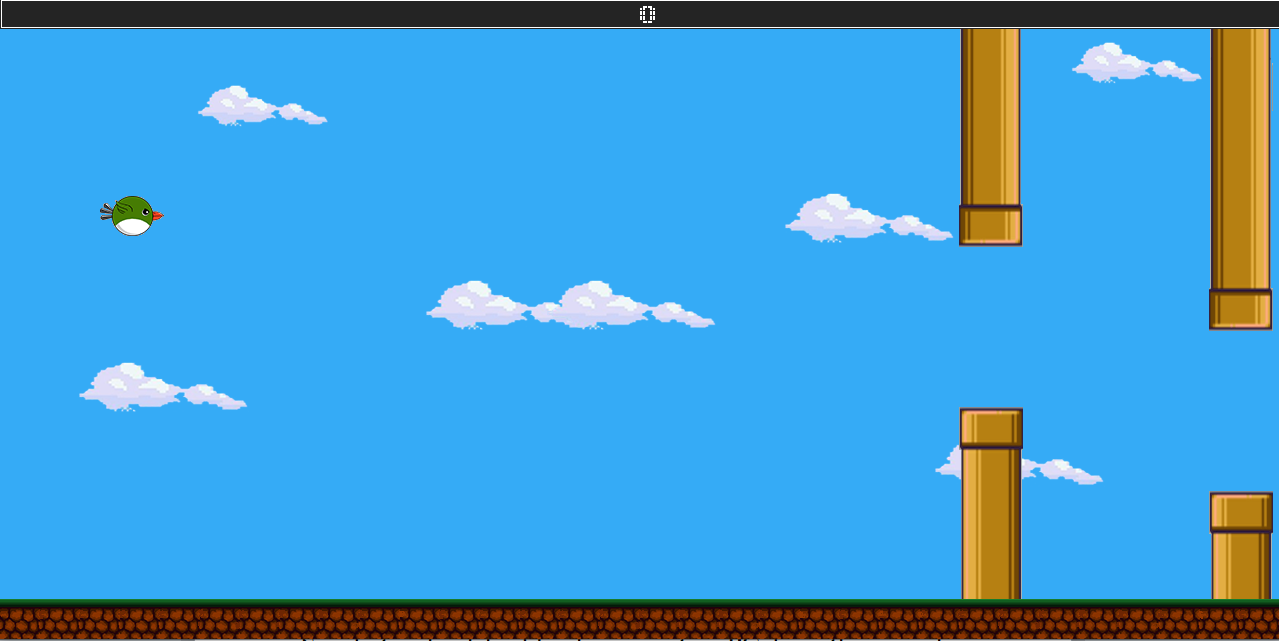
Với cùng một asset, em đã hoàn thành được game Flappy Bird trên cả engine tự xây dựng lẫn unity engine.

4.1.1 Giao diện Menu Game



**Hình 73** Menu Game

4.1.2 Giao diện Game Play



**Hình 74** Game play

4.1.3 Giao diện Game Over



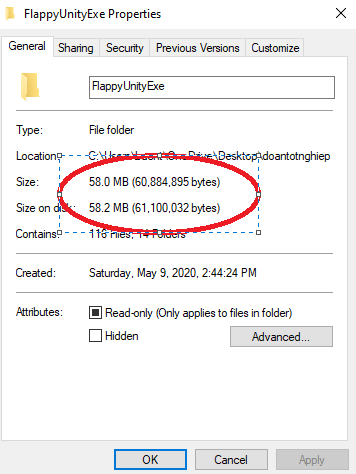
**Hình 75** Game Over

4.2 So sánh các chỉ số của 2 engine

Tiến hành build cả hai game trên PC để có thể so sánh về dung lượng, performance và các chỉ số liên quan khác.

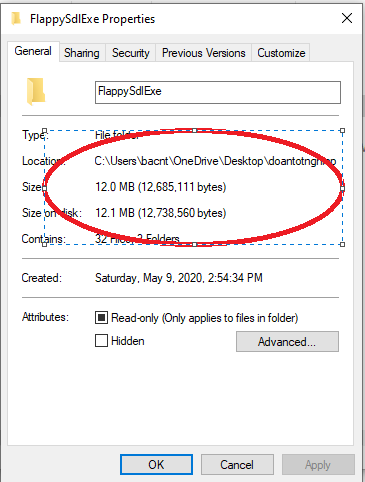
4.2.1 So sánh size build

4.2.1.1 Size build game với Unity Engine



**Hình 76** Thông số của file build Unity

4.2.1.2 Size build với Engine tự xây dựng



**Hình 76** Thông số của file build Engine tự viết

4.2.1.3 Đánh giá

Với cùng một asset giống nhau nhưng size build của Unity Engine là 58MB, còn size build của Engine tự xây dựng là 12MB. Điều đó cho thấy việc phát triển game dựa trên engine tự xây dựng sẽ tối ưu được rất nhiều về kích cỡ của dự án. Các thư viện thừa, các tính năng thừa và các tệp đính kèm không cần thiết trong dự án sẽ được loại bỏ hết, giúp cho hệ thống mở rộng tốt hơn và dễ dàng bảo trì hơn.

4.3 KẾT LUẬN

Trong suốt quá trình học hỏi và nghiên cứu đồ án tốt nghiệp, em đã hoàn thành sản phầm game Flappy Bird trên hai engine game cũng như xây dựng engine riêng với mã nguồn mở SDL vs mẫu ECS. Qua đó cũng đã nắm bắt được các vấn đề khi sử dụng các engine có sẵn, cũng như các kiến thức về thiết kế hướng dữ liệu.

Trong phạm vi đồ án và giới hạn về thời gian nên engine em phát triển vẫn còn những vấn đề và các hạn chế chưa giải quyết được hoàn toàn, engine xây dựng còn khá ít các chức năng và chưa áp dụng vào các dự án thực tế được.

Qua quá trình làm đồ án, dưới sự hướng dẫn tận tình của ThS. Lê Tấn Hùng và anh Hoàng Mạnh Hưng – Techcal Leader của công ty GameLoft, em đã học hỏi được rất nhiều kiến thức, tăng khả năng nghiên cứu, giải quyết vấn đề. Sắp tới, nếu có điều kiện em sẽ tiếp tục phát triển dự án này để có thể giải quyết được các vấn đề đang gặp phải.

TÀI LIỆU THAM KHẢO

[1.] S. Bilas, A Data-Driven Game Object System, Game Developer’s Conference, USA, 2002.

[2.] S. Bilas, A Data-Driven Game Object System, Game Developer’s Conference, USA, 2002, 41 pp., available: https://www.gamedevs.org/uploads/data-drivengame-object-system.pdf.

[3.]  [Brian Moakley](https://www.google.com/search?sa=X&biw=1872&bih=963&q=unity+games+by+tutorials:+make+4+complete+unity+games+from+scratch+using+c%23+brian+moakley&stick=H4sIAAAAAAAAAE3LPQoCMRBA4Uqw0cLaYtDOJiwsFrmJVcjGbBLyM8tkYlyP40kVbCwffG-7P-yEE8Ngruqh5mU8_fL-jJ1ezl-O0UgxIUbRKTDbojpSlLqxR3pvbq0EXsHpbCtMK3BjpKBTlZB1tDCCwbwkyxb-5UyYoRrSbDy0GooDc4bpexbIqGOy6wfPLnMDmgAAAA&ved=2ahUKEwjzgZShxsngAhWSMN4KHbkWC0AQmxMoATATegQIBRAH), [*Mike Berg*](https://www.google.com/search?sa=X&biw=1872&bih=963&q=unity+games+by+tutorials:+make+4+complete+unity+games+from+scratch+using+c%23+mike+berg&stick=H4sIAAAAAAAAAE3LsQrCMBCA4Ulw0cHZ4dDNJRSKSN7DOSTXNA1pcuVysfR1fFIdHBx_-P798XRQQXUd3s3LjEt_-WXv3KMMw-2cUCtHlNTKUcQXsxInbZtMxO_ds5UoGwSbfQW3gTQhjnauGrJNHnpAysvsxcO_HJkyVGQrOEGrsQTAK-T4PZzn8AGOjbjElgAAAA&ved=2ahUKEwjzgZShxsngAhWSMN4KHbkWC0AQmxMoAjATegQIBRAI) *- Unity Games by Tutorials: Make 4 Complete Unity Games from Scratch*, 2016.

[4.] https://www.gamedev.net/tutorials/\_/technical/game-programming/swept-aabb-collision-detection-and-response-r3084/

[5.] [Dieter Schmalstieg](https://www.google.com/search?sa=X&biw=1872&bih=963&q=augmented+reality:+principles+and+practice+dieter+schmalstieg&stick=H4sIAAAAAAAAAA3HsQrCMBAA0EkQFAdnh-DocraTBPyWck2O5EiaNJcrxd_xS_Vt73i5niHAMM6taWx6P8ECz2FM0yvFxy05C3OtCXZhVSrTXiVZ3DRW-R7euIWFipI3QphZP9aswsXxmqkbLP5fdMqOjGdSEtNdXDB3ZQo_NpT7MnsAAAA&ved=2ahUKEwiCyKqKx8ngAhXZ7GEKHW6pA_YQmxMoATASegQIBhAH), [Tobias Höllerer](https://www.google.com/search?sa=X&biw=1872&bih=963&q=augmented+reality:+principles+and+practice+tobias+h%C3%B6llerer&stick=H4sIAAAAAAAAAB3HOw7CMAwA0AmJAXVgZogYu0QwhsMgx7UaK2k-jkvFdTgEF-BiIN729sPxYGd7ufrWNDQ9_xew9seS6niK6KwvJdpNWJXyfSsSHawairx2N1jnhbLSZIQgsT6dqcIZuSbqBvL0K6AyktHiGboJn3dKJCRfeYpEcnoAAAA&ved=2ahUKEwiCyKqKx8ngAhXZ7GEKHW6pA_YQmxMoAjASegQIBhAI), *Augmented Reality: Principles and Practice*, 2015.

[6.] [Edward Lavieri](https://www.google.com/search?sa=X&biw=1872&bih=963&q=getting+started+with+unity+2018+-+third+edition:+a+beginner%27s+guide+to+2d+and+3d+game+development+with+unity,+3rd+edition+edward+lavieri&stick=H4sIAAAAAAAAAE3MMQ6CQBBG4ViY2GhhbfHHxsRoEAjRcBmyuMMysuyaZQC5gefwpJpYaPXyNW-xWi8jE8Vxde5sVhe37ZdlZqeqelT7TXPNo9L7JhoDi5ArRh-aXPVS-_CaP2eGRNgZdKKCkMbIUqN3LBOSU3zBEVJz0CDNwt7lUCjJsHMUdh1Mz5ogHomGchqphlEtQdNA1t9bcvJ3PCD9nT4d1YdWDUyB3zmCyJHKAAAA&ved=2ahUKEwiZnJW8xsngAhVGAogKHRUlDb0QmxMoATAPegQIBBAH), *Getting Started with Unity 2018 - Third Edition: A Beginner's Guide to 2D*, 2018.

[7.] https://www.libsdl.org/

[8.] T. Leonard, Postmortem: Thief: Dark Project, 9 July 1999 [Online], available (referred to 11.2.2019): http://www.gamasutra.com/view/feature/131762/postmortem\_thief\_the\_dark\_project.php.

[9.] https://entity-systems.wikidot.com/

[10.] https://www.azurefromthetrenches.com/introductory-guide-to-aabb-tree-collision-detection/

[11.] R. Lord,”Why use an Entity Component System architecture for game development?” 16 February 2012. [Online]. Available (referred to 15.2.2019): https://www.richardlord.net/blog/ecs/why-use-an-entity-framework.html.