|  |
| --- |
|  |

**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**ĐỒ ÁN TỐT NGHIỆP**

**Xây dựng mô-đun 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 cảm ơn

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 công ty TNHH Gameloft Việt Nam và ban giám hiệu nhà trường 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 sát cánh 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 mô-đun engine game sử dụng mã nguồn mở cung cấp các thư viện đồ họa, âm thanh là SDL2 và sử dụng mẫu thiết kế 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 mô-đun 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

**Lời cam kết** 2

**Lời mở đầu** 3

**Tóm tắt báo cáo** 4

**Mục lục** 5

**Danh mục hình vẽ** 7

**Chương 1 Giới thiệu về đề tài** 10

1.1 Đặt vấn đề 10

1.2 Mục tiêu và phạm vi của đề tài 11

1.2.1 Mục tiêu của đề tài 11

1.2.2 Phạm vi đề tài 11

1.3 Định hướng giải pháp 11

1.3.1 Mô tả bài toán 11

1.3.2 Định hướng 12

1.4 Bố cục đồ án 12

**Chương 2 Tổng quan về Game Engine và công nghệ sử dụng** 13

2.1 Game Composition 13

2.2 Game Engine 15

2.2.1 Kiến trúc của GameEngine 15

2.2.2 Platform Independence Layer 17

2.2.3 Utility Layer 17

2.2.4 Resource Management (Assets) 18

2.2.5 Game Engine Functionalities 18

2.2.5.1 Rendering Engine 19

2.3 Game Play Foundation 20

2.3.1 Game State 20

2.3.2 Game Flow 23

2.4 Các lớp cụ thể trong trò chơi 24

2.4.1 Physics Mô-đun 25

2.4.2 Collision Detection Basics 26

2.5 ECS 26

2.6 SDL 28

2.6.1 Giới thiệu về SDL 28

2.6.2 Các khái niệm cơ bản trong SDL 29

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

2.7.1 Khởi tạo cửa sổ 33

2.7.2 LoadTexture 33

2.7.3 Kỹ thuật Color Key 34

2.7.4 Sự kiện trong SDL 34

2.7.5 Bắt sự kiện nhấn hoặc thả ra. 35

2.7.6 Xử lý di chuyển cho đối tượng nhân vật 35

2.7.7 Xây dựng lớp Vector2D 36

2.7.8 Update, Render, Clean 37

2.7.9 Play Audio trong SDL 38

**Chương 3 Xây dưng mô-đun engine game và phát triển game Flappy Bird** 39

3.1 Xây dựng mô-đun engine game. 39

3.1.1 Xây dựng kiến trúc physics và collision mô-đun. 39

3.1.2 Phát triển mô-đun engine 39

3.1.2.1 Component 39

3.1.2.2 Entity 40

3.1.2.3 Manager 42

3.1.2.4 TransformComponent 43

3.1.2.5 SpriteComponent 44

3.1.2.6 KeyboardComponent 44

3.1.2.7 ColliderComponent 45

3.2 Phát triển game bằng engine tự xây dựng và Unity Engine 50

3.2.1 Giới thiệu về game Flapy Bird 50

3.2.2 Các đối tượng trong game. 50

3.2.3 Phát triển game bằng engine tự xây dựng. 53

3.2.4 Phát triển game bằng unity engine 54

3.2.4.1 Các thành phần có trong Unity Editor 54

3.2.4.2 Xây dựng game FlappyBird 58

**Chương 4 Các giải pháp và đóng góp** 61

4.1 Xây dựng mô-đun engine game 61

4.1.1 Giới thiệu bài toán. 61

4.1.2 Các vấn đề 61

4.1.3 Giải pháp 62

4.1.3.1 Sử dụng SDL 62

4.1.3.2 Thiết kế hướng dữ liệu và sử dụng ECS Pattern. 62

4.2 Kết quả đạt được 63

4.2.1 Size build game với Unity Engine 63

4.2.2 Size build với Engine tự xây dựng 64

**Chương 5 KẾT LUẬN** 66

5.1 Kết luận 66

**TÀI LIỆU THAM KHẢO** 67

Danh mục hình vẽ

[Hình 1 Tổng quan về game engine 14](#_Toc43679183)

[Hình 2 Tổng quan game engine 15](#_Toc43679184)

[Hình 3 Kiến trúc Game Engine 16](#_Toc43679185)

[Hình 4 Tổng quan về các lớp trong Game 17](#_Toc43679186)

[Hình 5 Kiến trúc Game Engine Functionalities 19](#_Toc43679187)

[Hình 6 Kiến trúc của Rendering Engine 19](#_Toc43679188)

[Hình 7 Các thành phần có trong Game State 21](#_Toc43679189)

[Hình 8 Phân phối Game State 22](#_Toc43679190)

[Hình 9 Mô hình Game Loop đơn giản nhất 23](#_Toc43679191)

[Hình 10 Game Flow 24](#_Toc43679192)

[Hình 11 Các thành phần vật lý trong game. 25](#_Toc43679193)

[Hình 12 Box Collider 26](#_Toc43679194)

[Hình 13 Bảng quan hệ Entity-Component. 27](#_Toc43679195)

[Hình 14 Hệ thống chuyển đổi về ECS 27](#_Toc43679196)

[Hình 15 Secret Maryo Chronicles 28](#_Toc43679197)

[Hình 16 7 Unknown Horizons 29](#_Toc43679198)

[Hình 17 Trine 29](#_Toc43679199)

[Hình 18 Sprite 30](#_Toc43679200)

[Hình 19 Sprite sheet 31](#_Toc43679201)

[Hình 20 Quy trình loading ảnh 32](#_Toc43679202)

[Hình 21 Template danh sách liên kết 33](#_Toc43679203)

[Hình 22 Load Texture trong SDL 33](#_Toc43679204)

[Hình 23 Load ảnh nhân vật lỗi 34](#_Toc43679205)

[Hình 24 Background nhân vật đã được xóa. 34](#_Toc43679206)

[Hình 25 Prototype của SDL\_PollEvent 35](#_Toc43679207)

[Hình 26 Xử lý bắt sự kiện trong SDL 35](#_Toc43679208)

[Hình 27 Xử lý di chuyển của đối tượng. 36](#_Toc43679209)

[Hình 28 Vector2D Class 37](#_Toc43679210)

[Hình 29 Các hàm cập nhật trong game 38](#_Toc43679211)

[Hình 30 Mô hình ECS cơ bản cấu thành nên engine 39](#_Toc43679212)

[Hình 31 Class Component 40](#_Toc43679213)

[Hình 32 Class Entity 41](#_Toc43679214)

[Hình 33 Các thành phần trong Entity 42](#_Toc43679215)

[Hình 34 Class Manager 43](#_Toc43679216)

[Hình 35 Class Transform Component 44](#_Toc43679217)

[Hình 36 Class SpriteComponent 44](#_Toc43679218)

[Hình 37 Class KeyboardComponent 45](#_Toc43679219)

[Hình 38 Class ColliderComponent 45](#_Toc43679220)

[Hình 39 Mô phỏng va chạm 46](#_Toc43679221)

[Hình 40 Mô phỏng va chạm giữa Object và Other 47](#_Toc43679222)

[Hình 41 Mô phỏng không va chạm giữa Object và Other 48](#_Toc43679223)

[Hình 42 Hiển thị các thông số cho Swept AABB 48](#_Toc43679224)

[Hình 43 Dự đoán va chạm của đối tượng với vật cản 50](#_Toc43679225)

[Hình 44 Menu Game 51](#_Toc43679226)

[Hình 45 Giao diện GamePlay 51](#_Toc43679227)

[Hình 46 Giao diện Game Over 52](#_Toc43679228)

[Hình 47 Đối tượng Bird 52](#_Toc43679229)

[Hình 48 Giao diện GamePlay 52](#_Toc43679230)

[Hình 49 Các thành phần cấu tạo nên Bird 53](#_Toc43679231)

[Hình 53 Cửa sổ scene trong Unity 55](#_Toc43679232)

[Hình 54 Cửa sổ Hierachy 56](#_Toc43679233)

[Hình 55 Cửa sổ Game 56](#_Toc43679234)

[Hình 56 Cửa sổ project 57](#_Toc43679235)

[Hình 57 Cửa sổ Inspector 58](#_Toc43679236)

[Hình 58 Thêm thành phần cho Bird 59](#_Toc43679237)

[Hình 59 Thêm collider cho Bird 59](#_Toc43679238)

[Hình 60 Load ảnh cho Sprite Renderer 60](#_Toc43679239)

[Hình 61 Hình ảnh đã được load trên scene 60](#_Toc43679240)

[Hình 62 Module Gameplay Foundations 61](#_Toc43679241)

[Hình 63 Hierarchy của OOC 62](#_Toc43679242)

[Hình 64 Hệ thống game Flappy Bird 63](#_Toc43679243)

[Hình 65 Thông số của file build Unity 64](#_Toc43679244)

[Hình 66 Thông số của file build Engine tự viết 65](#_Toc43679245)

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 mô đun engine riêng sử dụng mẫu thiết kế ECS và mã nguồn mở SDL2 nhằm giảm bớt các thư viện không cần thiết mà Unity Engine đính kèm để 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 mô-đun 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 mô-đun engine game riêng hơn là dùng một engine đã có sẵn.

Thể hiện được tổng quan các bước xây dựng các mô-đun 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 mô-đun engine game 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 mô-đun nhỏ rồi cấu hình cho từng mô-đun 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 mô-đun nhỏ đã xây dựng trước đó.

Sử dụng mã nguồn mở SDL2

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

1.4 Bố cục đồ án

Sau khi vấn đề, mục tiêu, phạm vi và định hướng giải pháp đã được xác định, các chương tiếp theo của đồ án được bố cục như sau.

Chương 2 trình bày tổng quan về Game Engine và các công nghệ sử dụng. Các công nghệ sử dụng gồm hai phần: công nghệ sử dụng trong xử lý đồ họa và công nghệ sử dụng cho thiết kế hệ thống.

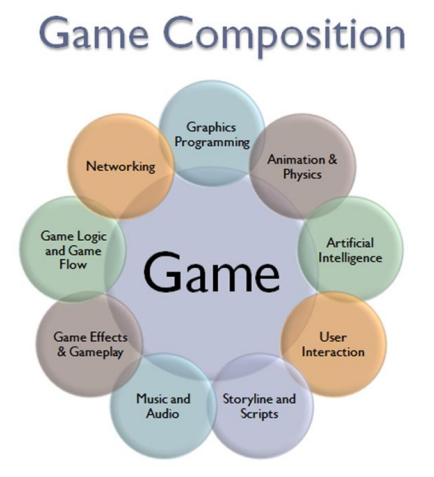
Chương 3 trình bày về phân tích thiết kế và xây dựng mô-đun engine.

Chương 4 trình bày về các đóng góp được đánh giá là tốt nhất cho hệ thống.

Cuối cùng, Chương 5 trình bày tổng hợp kết quả đạt được và hướng phát triển trong tương lai

Chương 2 Tổng quan về Game Engine và công nghệ sử dụng

2.1 Game Composition



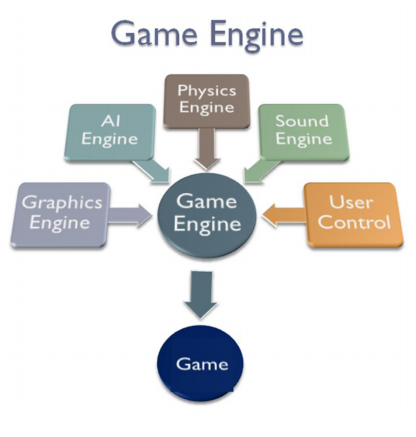
#### Hình 1 Tổng quan về game engine

Một engine game là một khung phần mềm được xây dựng, thiết kế và phát triển các trò chơi. Các nhà phát triển dùng chúng để tạo ra các trò chơi cho thiết bị di động và máy tính cá nhân.

Game Engine

* Bộ công cụ hỗ trợ xây dựng trò chơi: Buildbox, Game Market
* Thành phần phần mềm bất khả tri: cho phép tương tác giữa các hệ thống khác nhau
* Tái sử dụng: các mô-đun engine được viết ra với mục đích tái sử dụng cho việc xây dựng các trò chơi khác nhau.
* Đa nền tảng
* Cho phép phát triển hướng dữ liệu: phát triển hướng dữ liệu giúp cho việc đơn giản hóa mô hình.
* Làm cho sự phát triển nhanh hơn

2.2 Game Engine



#### Hình 2 Tổng quan game engine

Phần mềm được dùng để viết ra các trò chơi khác nhau trong một thể loại nhất định.

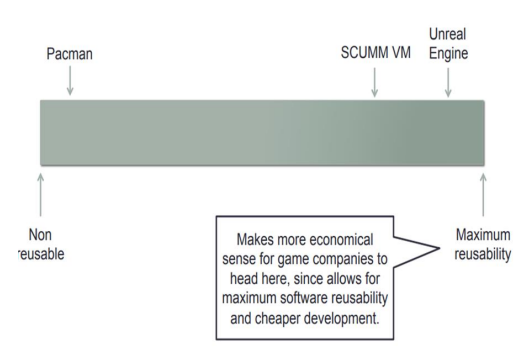
**Điển hình - hướng dữ liệu :**

* Logic trò chơi không được mã hóa vào mã nguồn, nhưng trong các tệp cấu hình bên ngoài.
* Hệ thống sẽ không phức tạp như OOP do không có kế thừa.

**Game vs Game-Engine:**

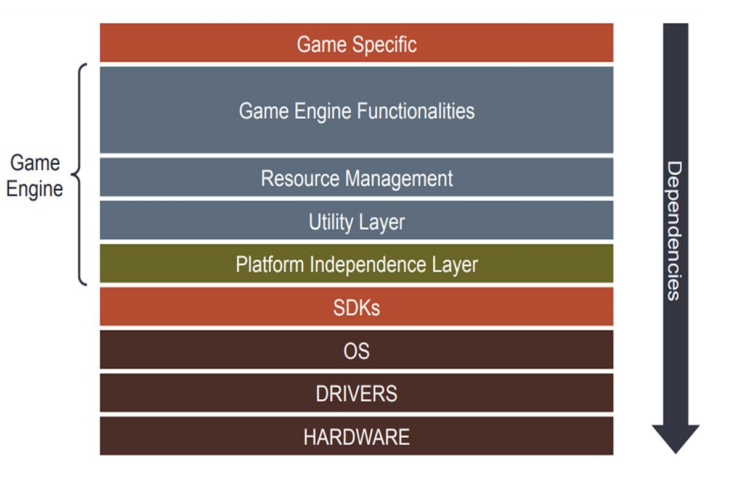
* Nếu một chương trình trò chơi được thay đổi để tái cấu trúc biến nó thành một trò chơi khác, thì đó là một trò chơi, nhưng không phải là một engine game.
* Nếu một chương trình trò chơi có thể thay đổi để chạy các trò chơi khác nhau một cách dễ dàng, thì đó là một công cụ trò chơi

2.2.1 Kiến trúc của GameEngine



#### Hình 3 Kiến trúc Game Engine

* Bao gồm một tập hợp các lớp, các lớp cung cấp các chức năng cấp cao hơn và cao hơn (không giống như một hệ điều hành thông thường)
* Có thể rất đơn giản và cũng có thể rất phức tạp.
* Ngoài lúc run-time (mã chạy khi trò chơi đang được chơi), công cụ trò chơi thường bao gồm các công cụ soạn thảo nội dung như asset game (trình chỉnh sửa cấp độ, trình chỉnh sửa tập lệnh, v.v.)



#### Hình 4 Tổng quan về các lớp trong Game

2.2.2 Platform Independence Layer

Đây là lớp cung cấp chức năng của tất cả các thư viện bên dưới một cách độc lập với nền tảng.

Ví dụ:

* Nếu ta mã hóa một trò chơi cho 2 nền tảng khác nhau, trong đó một nền tảng chỉ cho phép Physx và nền tảng kia là ODE, thì lớp này sẽ xác định API mà công cụ trò chơi có thể sử dụng, không ảnh hưởng và độc lập với thư viện vật lý bên dưới

Có nhiều lớp cung cấp sẵn đồ họa, âm thanh, v.v. phổ biến nhất là SDL (Simple Directmedia Layer) với :

* Đồ họa, âm thanh, bàn phím, chuột, mạng, luồng, v.v.
* Cung cấp các API để truy cập đến âm thanh, hình ảnh, v.v.

2.2.3 Utility Layer

Lớp này là tập hợp các chức năng cơ bản, không được cung cấp bởi bất kỳ SDK nào.

Điển hình là:

* Quản lý bộ nhớ: Cấp phát bộ nhớ động, tĩnh, ngoài ra cấp phát bộ nhớ tiêu chuẩn có thể không đủ nhanh cho công cụ trò chơi nhưng có thể tùy chỉnh được.
* Toán: tập hợp các hàm cơ bản cho hình học, lượng giác, bậc bốn, ma trận, giải phương trình, tích phân số, v.v. khi cần thiết cho công cụ trò chơi.
* Cấu trúc dữ liệu và giải thuật: cây, biểu đồ, thuật toán tìm kiếm, sắp xếp, v.v.
* Tạo số ngẫu nhiên
* Trình phân tích cú pháp tệp (XML, Json v.v.)
* Cấu hình

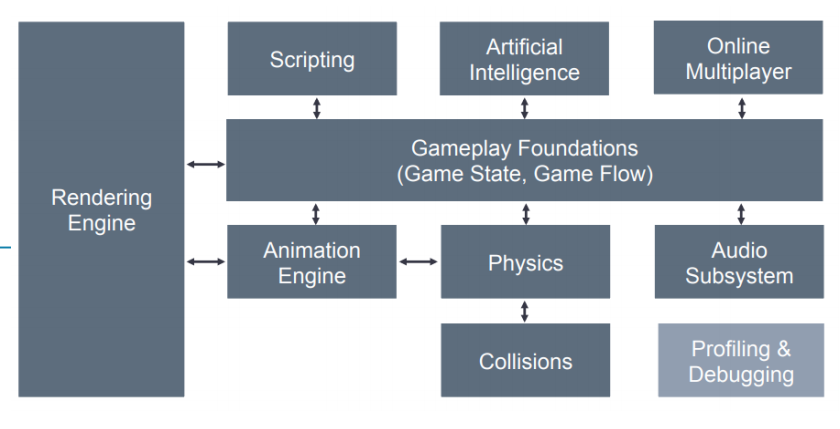
2.2.4 Resource Management (Assets)

Tất cả các trò chơi yêu cầu phải có asset (bitmap, mô hình 3D, script, hiệu ứng, bản đồ, âm thanh, v.v.)

Công cụ trò chơi sẽ cung cấp chức năng để tải, lưu, tạo, truy vấn vào asset, chẳng hạn như:

* Mô hình 3D: Kết cấu lưới / Vật liệu /Bộ xương Hoạt hình
* Mô hình 2D: animation Bitmap Skeletons, animation của đạn bay
* Phông chữ : các quốc gia có ngôn ngữ khác nhau nên phải có nhiều font chữ nếu muốn làm đa ngôn ngữ trong game.
* Âm thanh và âm nhạc : Âm nhạc cho các màn khác nhau hay âm thanh cho các nút button, hành động của nhận vật, v.v.
* Tài nguyên va chạm (hình dạng va chạm) : Khi va chạm phải có hiệu ứng va chạm hay các trạng thái animation khác nhau.

2.2.5 Game Engine Functionalities

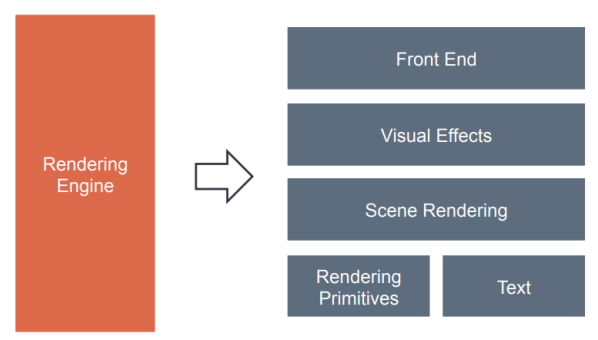


#### Hình 5 Kiến trúc Game Engine Functionalities

Đây là lớp phức tạp nhất của game engine vì nó bao gồm rất nhiều hệ thống con.

Các hệ thống con này sẽ đảm nhận các nghiệp vụ khác nhau để hình thành nên một trò chơi.

2.2.5.1 Rendering Engine



#### Hình 6 Kiến trúc của Rendering Engine

Vẽ đồ họa của trò chơi, đây là thành phần thiết yếu trong game gồm rất nhiều các mô-đun con. Nó có thể rất phức tạp nên cũng được chia thành một tập hợp các lớp con:

* Trình kết xuất cấp thấp: vật liệu, ánh sáng, máy ảnh, văn bản, nguyên thủy, chế độ xem, v.v.
* Vẽ cảnh : phân vùng cảnh
* Hiệu ứng hình ảnh: hệ thống material, shader, hiệu ứng particle system, v.v.
* Front end: HUD, GUI, menu, v.v

2.3 Game Play Foundation

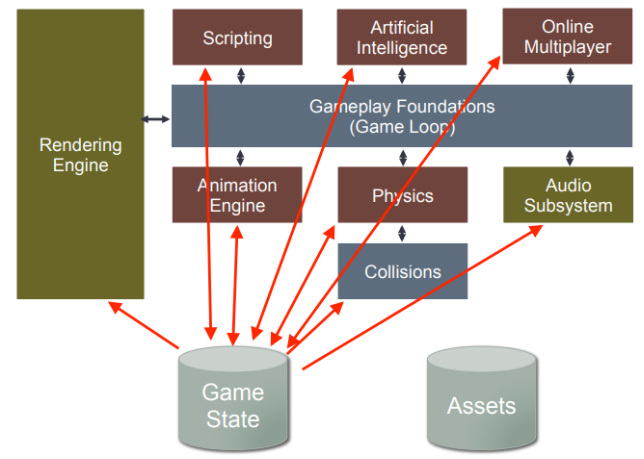
2.3.1 Game State

Trạng thái trò chơi: tập hợp các biến và cấu trúc dữ liệu cùng nắm bắt trạng thái hiện tại của trò chơi:

* Trạng thái bản đồ : Các đối tượng trên bản đồ đều cập nhật liên tục.
* Đối tượng (trạng thái vật lý) : đây là các thực thể trong game như nhân vật, vật cản, v.v.
* Nhân vật (trạng thái vật lý, trạng thái AI)
* Người chơi (trạng thái vật lý, hàng tồn kho, số liệu thống kê, v.v.)
* Camera : cho phép người dùng nhìn thấy một phần cảnh trong game

Những thứ KHÔNG phải là một phần của trạng thái trò chơi:

* Số điểm, cấu hình (độ phân giải, thiết lập bàn phím, v.v.), trạng thái tạm dừng, thua cuộc, v.v.
* Công cụ kết xuất đưa trạng thái trò chơi và hiển thị nó ra màn hình
* Các mô-đun khác trong trạng thái trò chơi cập nhật công cụ trò chơi (vật lý, AI) Điều rất quan trọng là trạng thái trò chơi được xác định rõ ràng



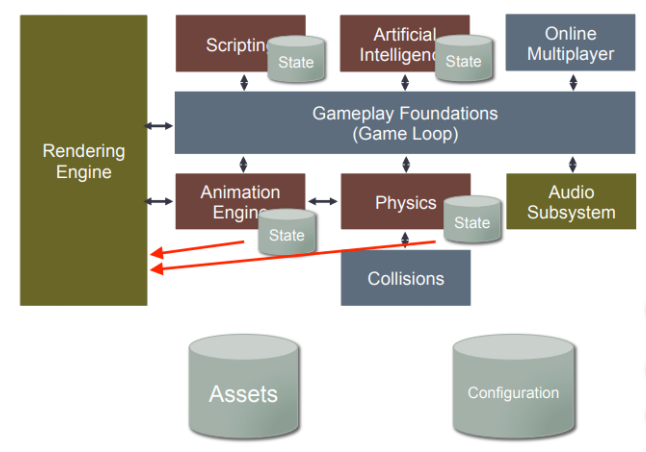
#### Hình 7 Các thành phần có trong Game State

Các cấu trúc dữ liệu hoặc cơ sở dữ liệu trong trạng thái trò chơi hoàn chỉnh

* Các mô-đun khác không cần can thiệp mà chỉ cần đọc hoặc cập nhật nó.
* Các mô đun không có trạng thái nội bộ với nhau.

Trong một Game Engine:

* Tách các thành phần ra khỏi phần còn lại của mã
* Tạo ra các mô-đun mạng riêng biệt



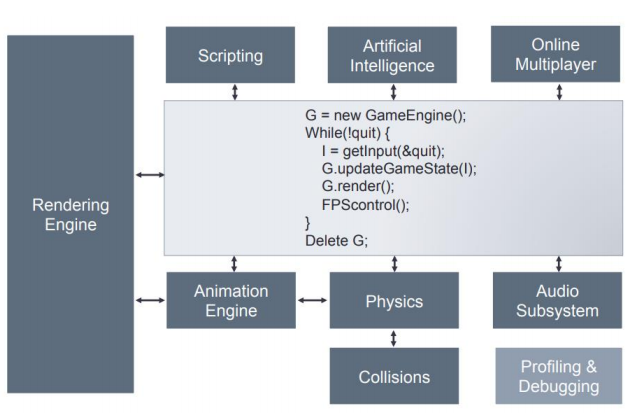
#### Hình 8 Phân phối Game State

* Tùy chọn thực tế (điển hình nhất): trạng thái của trò chơi được phân phối giữa các mô-đun.
* Dễ dàng mã hóa hơn nếu các trạng thái được phân tán (nhưng sau đó có thể gây đau đầu hơn cho việc lưu / tải trạng thái trò chơi hoặc mạng)
* Một Trạng thái Trò chơi phân tán yêu cầu một thiết kế rất có cấu trúc gọn và dễ tái sử dụng của các lớp trò chơi, để chúng cung cấp một giao diện đẹp cho mã mạng, ví dụ, để cập nhật / đọc trạng thái

In game:

* Nó không thực sự quan trọng, do mỗi trò chơi lại có một cách chơi khác nhau nên rất có khả năng nó sẽ không được sử dụng lại trong tương lai.

**Game Loop**



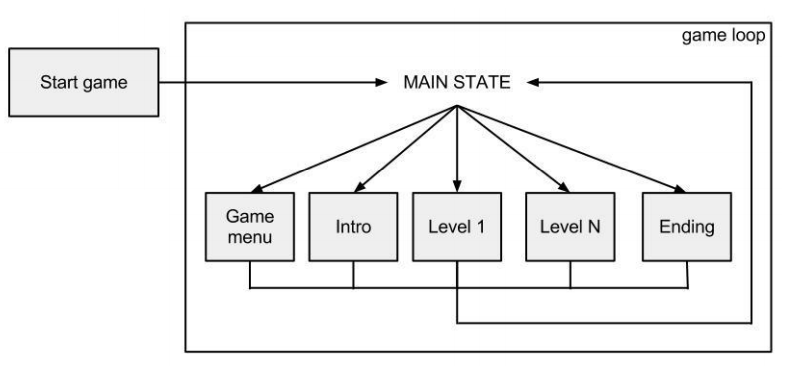
#### Hình 9 Mô hình Game Loop đơn giản nhất

Các vòng lặp trò chơi thường có một luồng nhưng qua quá trình phát triển, trò chơi đa luồng đang dần cải thiện các vấn đề của đơn luồng:

* Chạy từng hệ thống con trong một luồng
* Một luồng để kết xuất liên tục
* Một luồng để cập nhật vật lý liên tục
* Một luồng cho AI

Chúng là chế độ phức tạp, cần đồng bộ hóa vì dễ bị lỗi hơn và khó gỡ lỗi.

2.3.2 Game Flow



#### Hình 10 Game Flow

Cốt lõi của công cụ trò chơi là xác định vị trí của Game Loop, cũng như duy trì trạng thái trò chơi. Vị trí của game flow trên FSM (chuyển từ menu sang trò chơi, sang trò chơi, v.v.)

• Game flow đề cập đến cách trò chơi chuyển từ các scene. Ví dụ: menu chính sang chơi trò chơi, hoặc tạm dừng hoặc sang trò chơi trên màn hình, v.v.

• Thường được các nhà phát triển triển khai bên ngoài công cụ trò chơi (nhưng có thể ở bên trong)

• Công cụ trò chơi chỉ thực hiện phần chơi của trò chơi

• Phải xây dựng các chức năng thực hiện chuyển các menu, màn hình trò chơi, màn hình cấu hình bàn phím, v.v.

• Mỗi trò chơi thực hiện theo một cách khác nhau.

2.4 Các lớp cụ thể trong trò chơi

Engine Game thường cho phép dễ dàng cắm thêm các mô-đun dành riêng cho trò chơi. Ví dụ:

* Physic và Va chạm (trên Unity, khi khởi tạo Game Object mặc định phải có Transform, ngoài ra có thẻ thêm các collider để khám phá ra va chạm của nó với các Game Object khác)
* Tệp cấu hình cho các đơn vị trò chơi như phương tiện, vũ khí, nhân vật hay các hiệu ứng trò chơi.
* Các mô-đun hiệu ứng hình ảnh bổ sung, như các plug-in, tool render
* Kịch bản bổ sung và cấu hình hành vi cho nhân vật hoặc người chơi chính.
* AI (AI cho nhân vật hay cho Enemy)
* Cấu hình camera (cho người cho nhìn thấy một phần hay tất cả màn hình chơi)

2.4.1 Physics Mô-đun



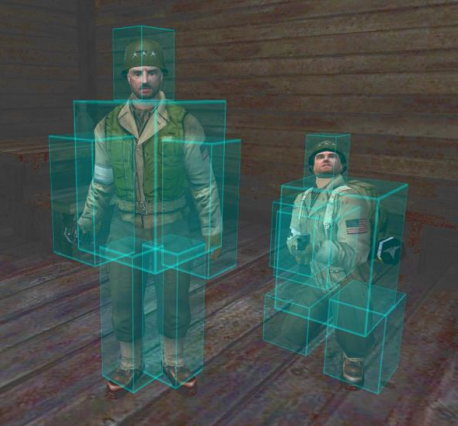
#### Hình 11 Các thành phần vật lý trong game.

Mô-đun vật lý duy trì các đối tượng trò chơi trong đó chúng có tọa độ, va chạm hay các biến đổi về trạng thái (những đối tượng chịu sự vật lý):

* Duy trì thông tin liên quan đến vật lý: hình dạng, khối lượng, vận tốc, gia tốc, v.v.
* Các đối tượng này là các thành phần chính cấu tạo nên một trò chơi.

Danh sách các đối tượng đó có thể tách biệt với các đối tượng khác ( có thể ở màn menu hay màn thua cuộc) được duy trì bởi phần còn lại của công cụ trò chơi.

2.4.2 Collision Detection Basics



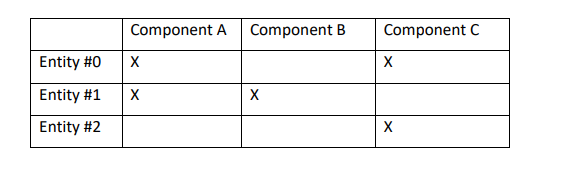
#### Hình 12 Box Collider

Mỗi đối tượng trò chơi (bao gồm cả nền) được liên kết với hình dạng va chạm, khi có các liên kết này, việc khám pha được va chạm sẽ thông qua tọa độ 2D hoặc 3D của các liên kết. Việc xử lý logic game chủ yếu thông qua việc kiểm tra và xử lý va chạm.

2.5 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.

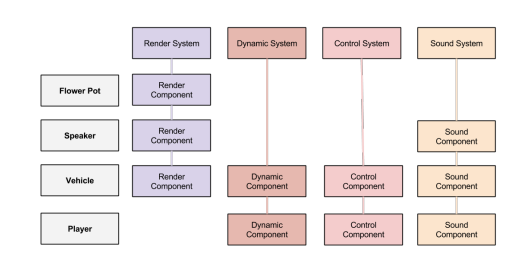
Lập trình một hệ thống với Entity-Component-System giống như lập trình hướng dữ liệu. Một thể hiện của thực thể hoạt động tương tự một khóa trong cơ sở dữ liệu, giống như bất kỳ hệ thống quản lý cơ sở dữ liệu quan hệ nào. Từ quan điểm trừu tượng, truy cập vào một thành phần của một thực thể cụ thể như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



#### Hình 13 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.

**Hệ thống ECS**



#### Hình 14 Hệ thống chuyển đổi về 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.

2.6 SDL

2.6.1 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 15 Secret Maryo Chronicles



#### Hình 16 7 [Unknown Horizons](https://vi.wikipedia.org/w/index.php?title=Unknown_Horizons&action=edit&redlink=1)



#### Hình 17 [Trine](https://vi.wikipedia.org/w/index.php?title=Trine_(Computerspiel)&action=edit&redlink=1)

2.6.2 Các khái niệm cơ bản trong SDL

**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.

**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

**Frame**

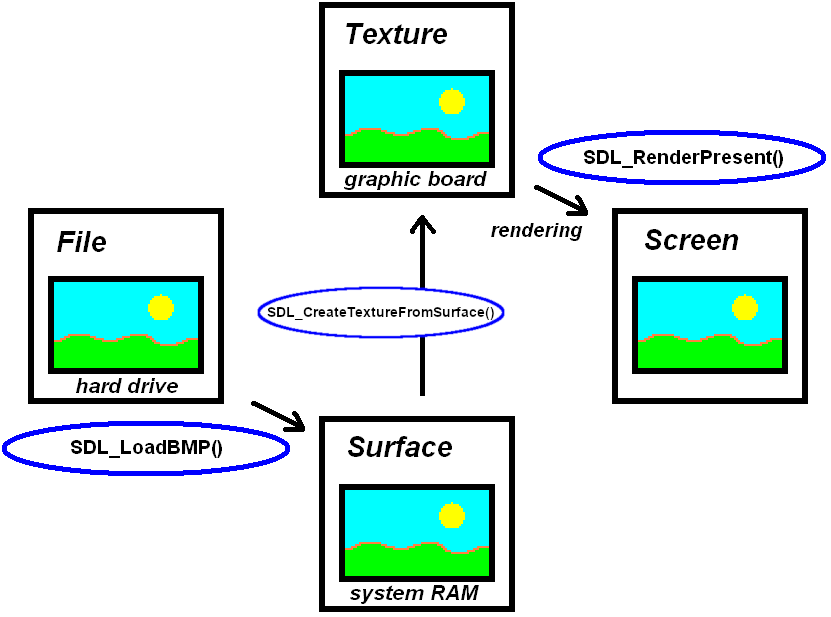
Tất cả các thiết bị hiển thị hình ảnh chuyển động hiện nay như TV, monitor… đều dựa trên khả năng lưu ảnh của mắt người. Sau khi hình ảnh biến mất mắt của chúng ta còn lưu lại tại vỏng mạc 1/24 giây. Vì vậy để tạo các đoạn film người ta chỉ cần thay đổi liên tục các hình ảnh theo tốc độ 24 hình/giây 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

**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.

**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.



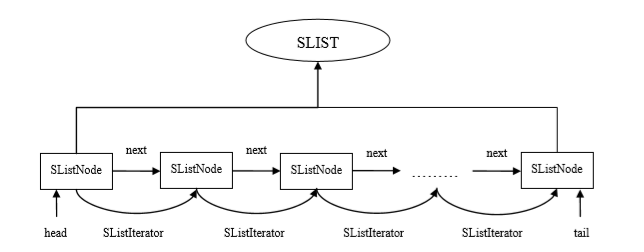
#### Hình 20 Quy trình loading ảnh

**Structure Containers**

Đây là kĩ thuật dùng để tích hợp các mô-đuns 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

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

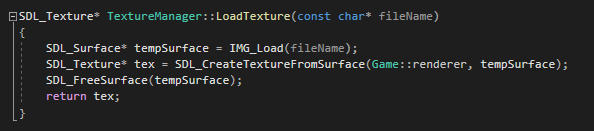
2.7.1 Khởi tạo cửa sổ

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.

2.7.2 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 22 Load Texture trong SDL

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

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).

2.7.3 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 23 Load ảnh nhân vật lỗi

Kỹ thuật color key giúp chúng ta xóa background. Với điều kiện mã màu truyền vào phải trùng với mã màu của background.



#### Hình 24 Background nhân vật đã được xóa.

2.7.4 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.

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:



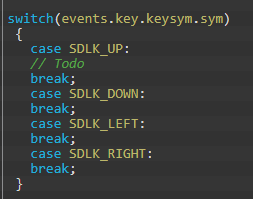
#### Hình 25 Prototype của SDL\_PollEvent

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.

2.7.5 Bắt sự kiện nhấn hoặc thả ra.

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

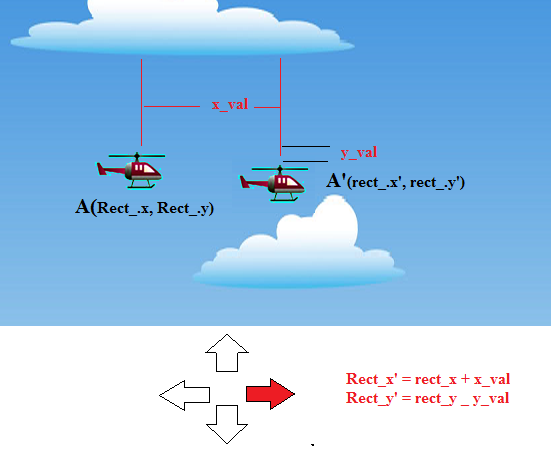
**Ví dụ ta muốn bắt sự kiện: Left-Right-Up-Down để di chuyển nhân vật.**



#### Hình 26 Xử lý bắt sự kiện trong SDL

2.7.6 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.



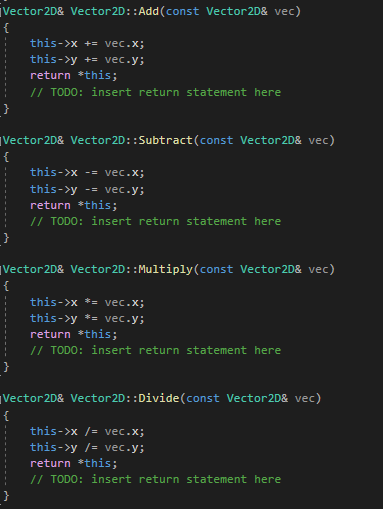
#### Hình 27 Xử lý di chuyển của đối tượng.

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.

2.7.7 Xây dựng lớp Vector2D

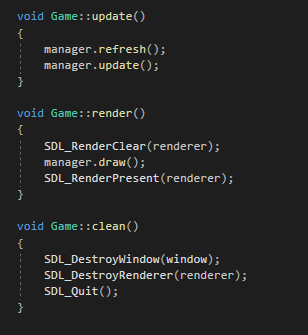
Nguyên lý cơ bản: Lớp Vector2D này chứa hai giá trị x và y là thể hiện của đối tượng trong không gian hai chiều.

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 28 Vector2D Class

2.7.8 Update, Render, Clean



#### Hình 29 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.

2.7.9 Play Audio trong SDL

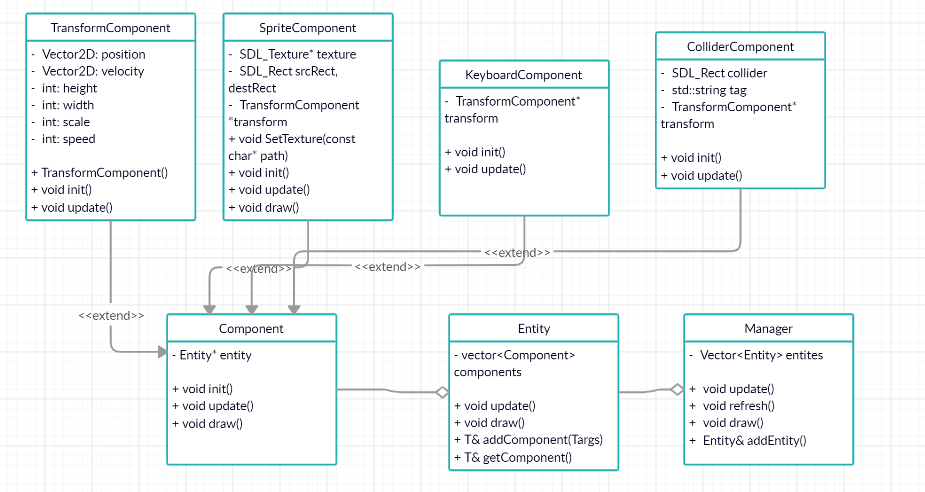
Tích hợp SDL\_mixer vào project, trước khi sử dụng function mà SDL\_mixer cung cấp. SDL\_mixer có hỗ trợ chúng ta hai cấu trúc để quản lý Music và Sound Effect là Mix\_music và Mix\_Chunk.

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.

Chương 3 Xây dưng mô-đun engine game và phát triển game Flappy Bird

3.1 Xây dựng mô-đun engine game.

3.1.1 Xây dựng kiến trúc physics và collision mô-đun.



#### Hình 30 Mô hình ECS cơ bản cấu thành nên engine

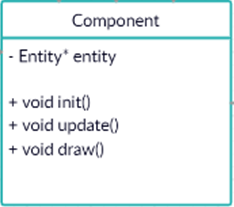
Mỗi một mô-đun được tạo nên đều có những thành phần cơ bản với các nghiệp vụ tương ứng và cần thiết cho mỗi đối tượng trò chơi, mà cụ thể ở đây là Transform Component, Sprite Component, Keyboard Component, Collider Component. Nhưng để có thể chia thành các mô-đun xử lý riêng ta cần một nền tảng cơ sở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

3.1.2 Phát triển mô-đun engine

3.1.2.1 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 31 Class Component

Thiết lập một Base Component trong mô-đun engine game. Gồm các hàm cơ bản như init(), update() hay draw().

3.1.2.2 Entity

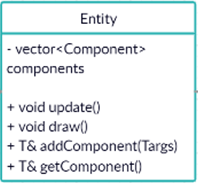
Tất cả đối tượng trong game đều là những thực thể (entity), và mỗi thực thể là duy nhất (có một UniqueID). Và những đặc điểm (property), trạng thái (state), phản ứng (behaviour) của các entity được quyết định bởi các Component.

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)

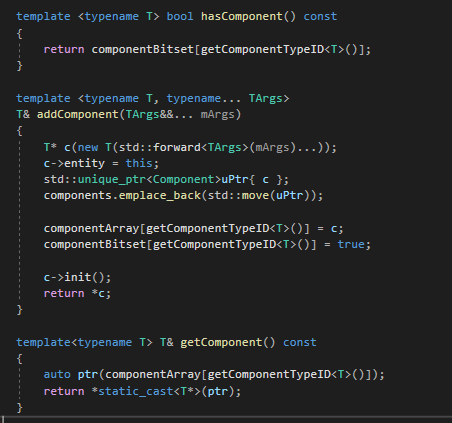
Pipe (ống vật cản) cũng là entity, nó bao gồm: Thành phần hiển thị (Renderable), yếu tố vật lý (Collider, Body), và không có yếu tố về di chuyển (Movable) và AI.



#### Hình 32 Class Entity

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.

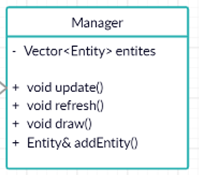


#### Hình 33 Các thành phần trong Entity

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 34** Class Manager

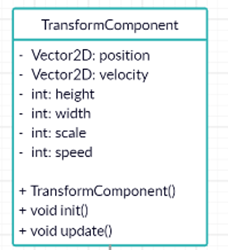
Manager đóng một vai trò quan trọng trong game. Nó như là đầu não của game, chứa tất cả các thực thể có trong game. Và việc các thực thể hoạt động ra sao, được cấu hình như nào thì đều thông qua class Manager.

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:

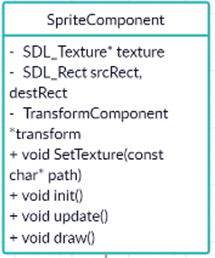
3.1.2.4 TransformComponent



#### Hình 35 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.

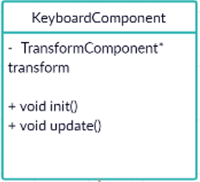
3.1.2.5 SpriteComponent



#### Hình 36 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.

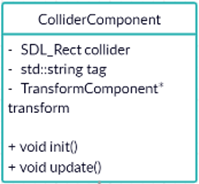
3.1.2.6 KeyboardComponent



#### Hình 37 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



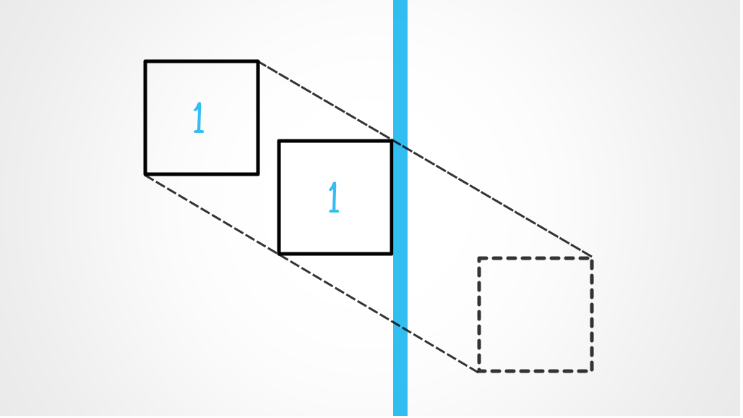
#### Hình 38 Class ColliderComponent

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.

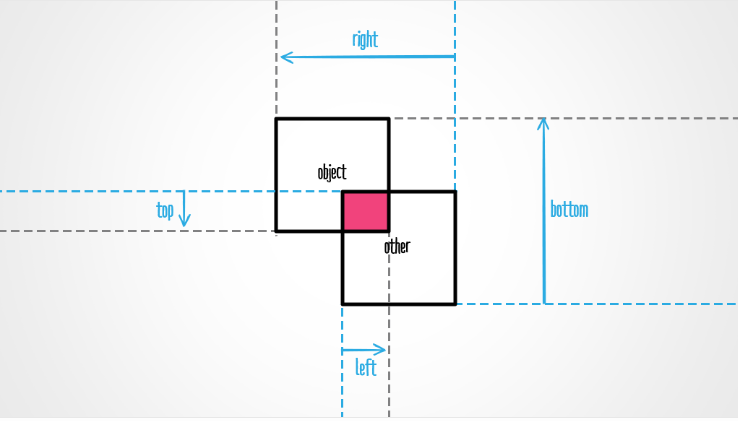
Kỹ thuật phát hiện va chạm dựa vào hộp bao AABB.

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 39 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 40 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**:

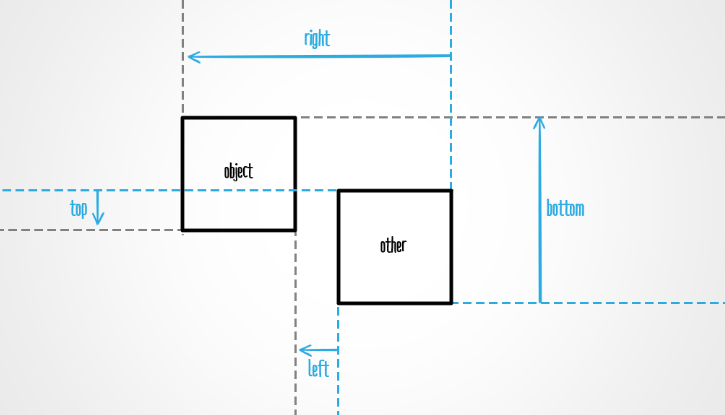
Other.left <= object.right

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.

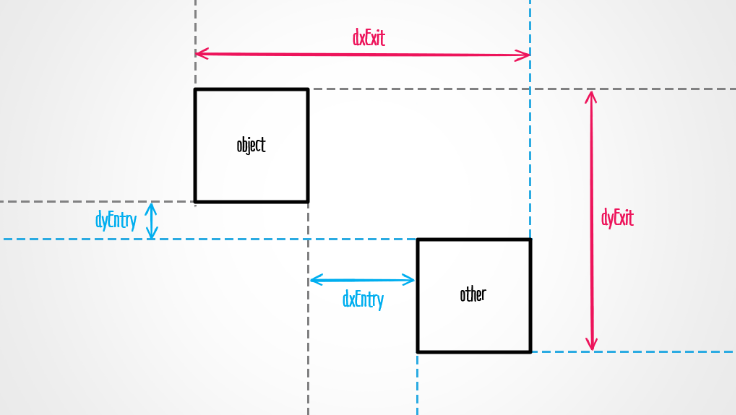


#### Hình 41 Mô phỏng không va chạm giữa Object và Other

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 42 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.

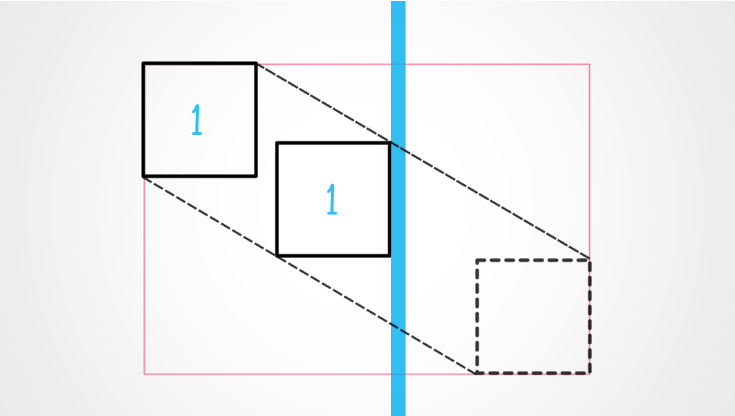
Thời gian để kết thúc va chạm nhỏ hơn 0.0f: 2 hình chữ nhật đang đi ra xa nhau.

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 43 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.

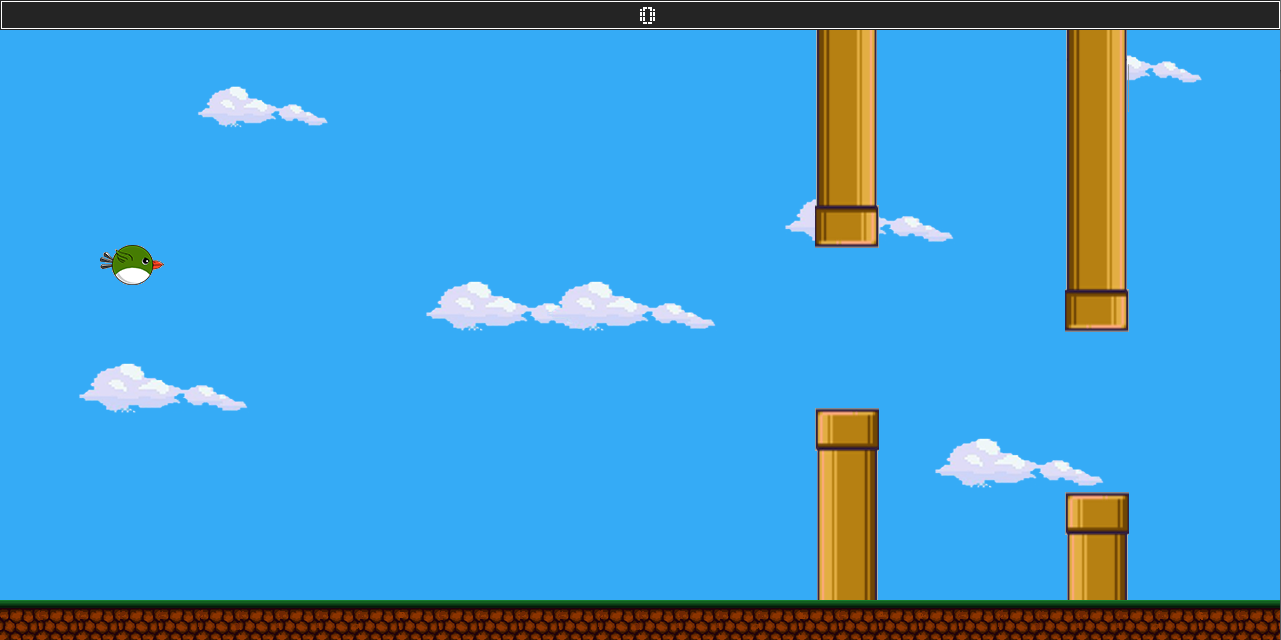
**Giao diện Menu Game**

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 44 Menu Game

**Giao diện game play**



#### Hình 45 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 46 Giao diện Game Over

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

**Đối tượng Bird**



#### Hình 47 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**



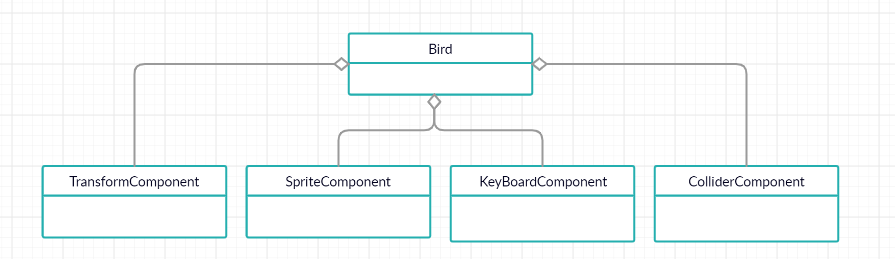
#### Hình 48 Giao diện GamePlay

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 49 Các thành phần cấu tạo nên Bird

Thực thể Bird sẽ được thêm 4 thành phần gồm:

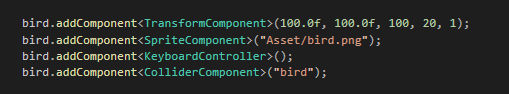
**TransformComponent**: xác định vị trí của Bird.

**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

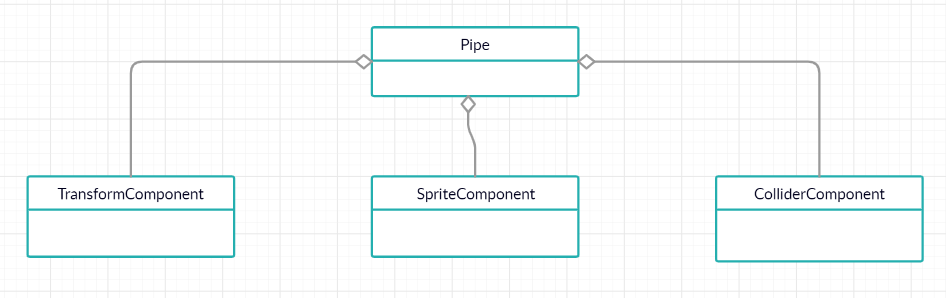
Bird sẽ được thêm các component như sau:

  
Hình 50 Cách thêm các thành phần vào Bird

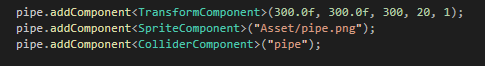
Khi thêm các component, ta cũng đồng thời thêm đầu vào cho Bird như vị trí, load sprite, thêm keyboard để người dùng thao tác, và bird tag để có thể xử lí va chạm.

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

Xây dựng thực thể Pipe:

Hình 51 Các thành phần cấu tạo nên Pipe

Pipe sẽ được thêm các component như sau:

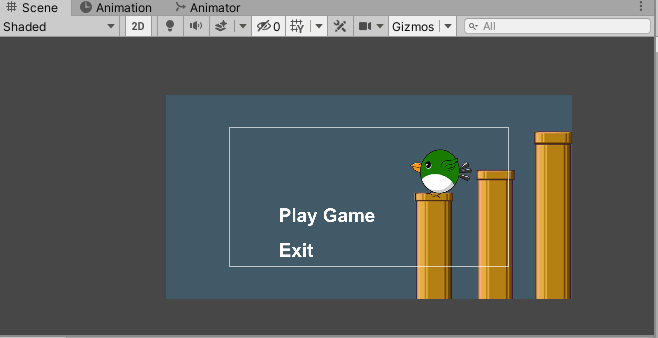
Hình 52 Cách thêm các thành phần vào Bird

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**

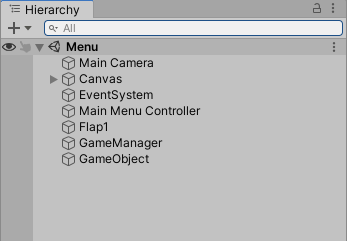


#### Hình 53 Cửa sổ scene trong Unity

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 54 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**

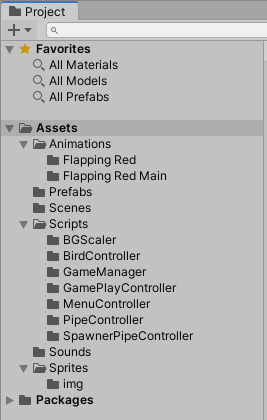


#### Hình 55 Cửa sổ Game

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 56 Cửa sổ project

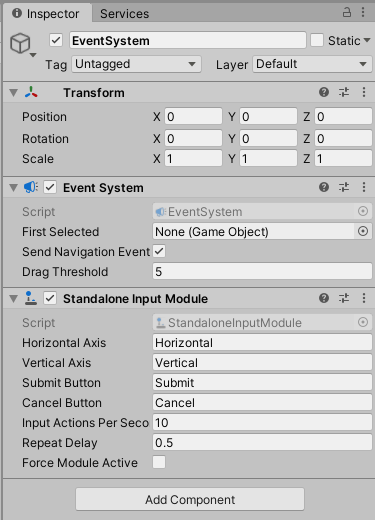
Đây là cưa sổ explorer của Unity, hiển thị thông tin của tất cả các tài nguyên (Assets) trong game của bạn.

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 57 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.

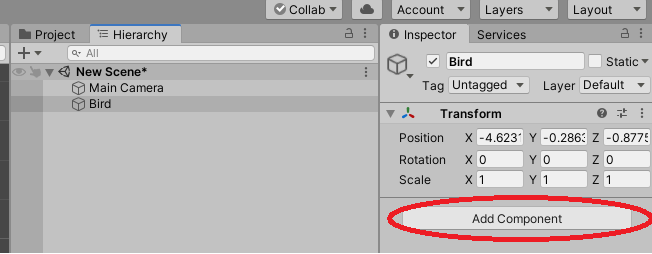
Mọi thuộc tính thể hiện trong Inspector đều có thể dễ dàng tuỳ chỉnh trực tiếp mà không cần thông qua một kịch bản định trước. Tuy nhiên Scripting API cung cấp một số lượng nhiều và đầy đủ hơn do giao diện Inspector là có giới hạn.

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, …

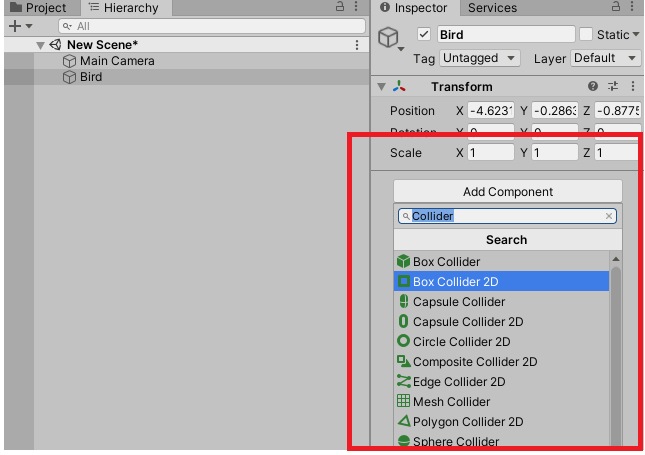
3.2.4.2 Xây dựng game FlappyBird

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



#### Hình 58 Thêm thành phần cho Bird

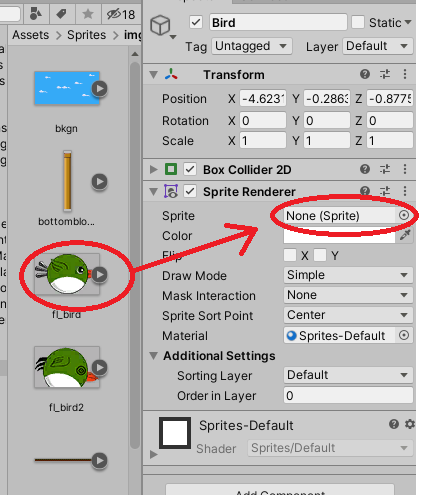
Khởi tạo một Game Object Bird, khi được sinh ra, Bird sẽ được mặc định có Transform Component. Do Unity đã cũng cấp sẵn các thư viện về Collider hay SpriteRender, nên ta chỉ việc AddComponent cho Bird.



#### Hình 59 Thêm collider cho Bird

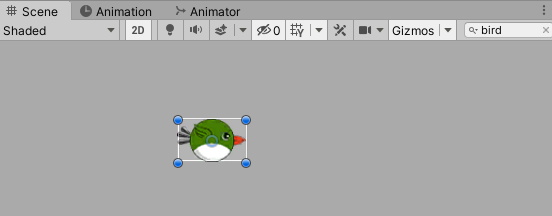
Unity cung cấp rất nhiều các thành phần xử lý va chạm cho cả 2D lẫn 3D. Ở đây chúng ta chọn BoxCollider2D.

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 60 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 61 Hình ảnh đã được load trên scene

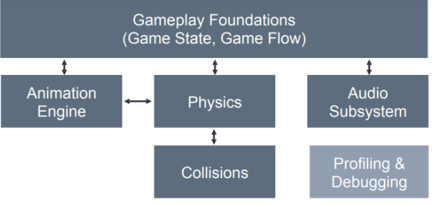
Chương 4 Các giải pháp và đóng góp

4.1 Xây dựng mô-đun engine game

4.1.1 Giới thiệu bài toán.

Unity là một game engine lớn, với rất nhiều tính năng, nền tảng nên nó đính kèm rất nhiều thứ cho tất cả các nhà phát triển sử dụng. Vì vậy có nhưng tệp dư thừa không cần thiết cho dự án game mà không thể tối ưu được. Điều đó dẫn tới ảnh hưởng đến hiệu năng game, kích cỡ của file build lớn, và crash game do tràn ram.

Để giải quyết vấn đề này, giải pháp đưa ra là xây dựng một engine game riêng để loại bỏ tất cả các phần thừa, không cần thiết cho dự án game. Tuy nhiên xây dựng được một engine game cần rất nhiều thời gian và một đội ngũ lớn mạnh, vì vậy trong phạm vi của đồ án, em đã tập trung phát triển mô-đun Physics và Collision trong phần Gameplay Foundations.



#### Hình 62 Module Gameplay Foundations

4.1.2 Các vấn đề

Việc xây dựng một mô-đun engine là làm mới hoàn toàn, nên có những khó khăn về việc thiết kế, tư tưởng lập trình, hay xử lý đồ họa.

* Xây dựng mô-đun engine game theo lập trình hướng đối tượng dẫn đến kiến trúc Game Object bị phức tạp.
* Khi mở rộng chức năng cho các đối tượng khác nhau thì không thể tái sử dụng lại mã nguồn.
* Tìm kiếm một lớp Platform Independence Layer

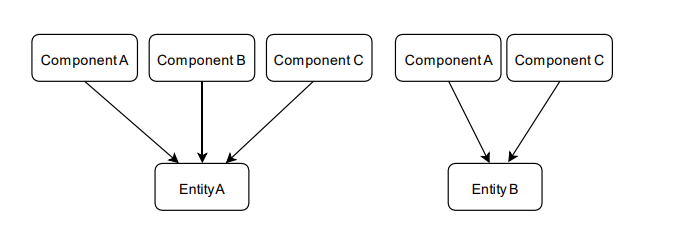
4.1.3 Giải pháp

4.1.3.1 Sử dụng SDL

Sử dụng thư viện SDL đóng vai trò như một Platform Independence Layer để cung cấp các API xử lý đồ họa, âm thanh.

4.1.3.2 Thiết kế hướng dữ liệu và sử dụng ECS Pattern.

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.

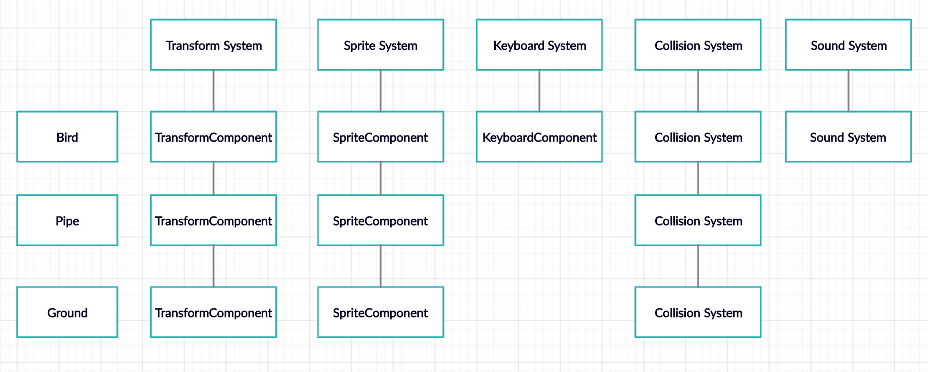


#### Hình 63 Hierarchy của OOC

Trong **Hình 63**, 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.

Sử dụng mẫu ECS Pattern để xây dựng mô-đun Physics và Collision.

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.



#### Hình 64 Hệ thống game Flappy Bird

Ưu điểm của ECS bao gồm:

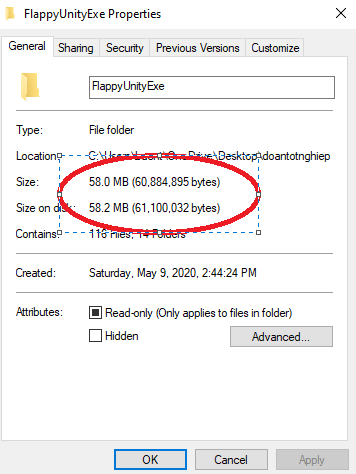
* Đảm bảo nguyên tắc thực hiện một nhiệm vụ duy nhất.
* Khả năng kết hợp hay định nghĩa đối tượng trong lúc runtime.
* Dễ dàng testing, với mỗi component hay system.
* Tính song song, có thể xây dựng nhiều System làm việc cùng một lúc.
* Phân tách dữ liệu và anh vi, có thể mở rộng các anh vi khác nhau trên cùng một dữ liệu.

4.2 Kết quả đạt được

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

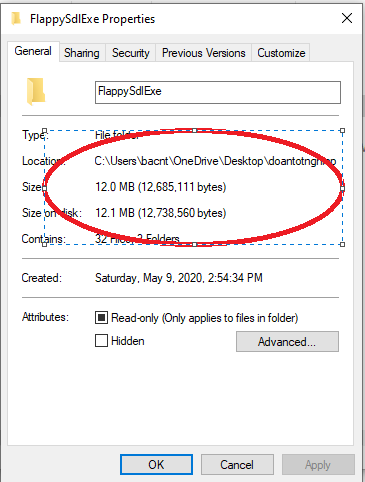
Tiến hành build cả hai game trên PC để có thể so sánh về dung lượng, performance và các chỉ số liên quan khác.

4.2.1 Size build game với Unity Engine



#### Hình 65 Thông số của file build Unity

4.2.2 Size build với Engine tự xây dựng



#### Hình 66 Thông số của file build Engine tự viết

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.

Chương 5 KẾT LUẬN

5.1 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 mô-đun engine riêng với mã nguồn mở SDL vs mẫu thiết kế 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 mô-đun engine em phát triển vẫn còn những vấn đề và các hạn chế chưa giải quyết được hoàn toàn, mô-đun 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 – Technical 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.