Đề tài: Lập trình game Run!!

**Lời mở đầu**

**Mục lục**

I. Giới thiệu về đề tài:

1. Tổng quan về đề tài:

Lập trình game là đã và đang là lĩnh vực đươc quan tâm và chú trọng hàng đầu trong ngành công nghiệp nội dung số. Với doanh số toàn cầu tăng trưởng chóng mặt, lập trình game đã trở thành ngành có tốc độ phát triển nhanh nhất trong lĩnh vực giải trí.

Ở Việt Nam, nghề lập trình game tuy đã xuất hiện từ rất lâu tuy nhiên chỉ ở những năm gần đây khi mà thị trường smartphone bùng nổ và đặc biệt là sự thành công của Nguyễn Hà Đông với tựa game Flappy Bird, nghề lập trình game được quan tâm hơn bao giờ hết.

Game Run!! được dựa trên ý tưởng của game Geometry Dash. Là một tựa game rhythm-based platformer, Game cho phép người chơi vào nhân vật hình vuông, sử dụng one-touch gameplay, người chơi điều khiển nhân vật qua các chướng ngoại vật đến khi về đích

**(Ảnh game)**

Ngoài ra Game Run!! còn hỗ trợ multiplayer với tối đa 4 người cùng chơi.

**(Ảnh Multiplayer trong game)**

2. Công nghệ sử dụng:

Android Studio: Là một môi trường phát triển tích hợp (IDE) cho phép người dùng phát triển các ứng dụng cho điện thoại android một cách nhanh chóng, dễ dàng.

LibGDX: Là một framework phát triển game cả 2D và 3D, được viết bằng ngôn ngữ lập trình Java. LibGDX cho phép phát triển game trên nhiều nền tảng khác nhau mà chỉ cần 1 lần viết code.

Tiled Map Editor: Là một công cụ giúp người dùng vẽ các map trong game (Đặc biệt là tile map) dễ dàng hơn.

Firebase: Là một dịch vụ cơ sở dữ liệu thời gian thực hoạt động trên nền tảng đám mây được cung cấp bởi Google nhằm giúp các lập trình viên phát triển nhanh các phần back-end của ứng dụng như Database, Hosting, Authentication, …

Node.js: Là một nền tảng mã nguồn mở cho phép xây dựng các ứng dụng liên quan đến network và quản lý server bằng ngôn ngữ JavaScript.

Socket.io: Là một thư viện chạy trên Node.js cho phép xây dựng dễ dàng hơn ứng dụng có sự giao tiếp 2 chiều giữa client và server.

3. Mục tiêu:

Ngoài mục đích học thêm về Android Studio và ngôn ngữ Java, việc xây dựng ứng dụng còn giúp nhóm học thêm về những khái niệm căn bản có trong game, cách game hoạt động với sự hỗ trợ của LibGDX framework. Việc xây dựng phần chơi multiplayer của game còn giúp nhóm có một khái niệm căn bản về việc xây dựng server-client.

II. Thiết kế:

1. Kiến trúc chương trình:

Game sẽ được chia các màn hình game được gọi là Screen và được quản lý bởi một lớp Game Manager duy nhất.

Các Screen cơ bản:

* Menu Screen
* Map Selection Screen
* Play Screen

Ngoài ra còn một số Screen khác tuy nhiên ở đây để phục vụ cho mục đích dễ dàng hiểu rõ hơn kiến trúc chương trình, chúng ta sẽ xét hoạt động của 3 Screen trên.

Cách thức game hoạt động và các Screen cơ bản được mô tả như hình bên dưới:

Các lệnh được gọi

Người dùng kết thúc game

Game được kết thúc

Bắt đầu game

Game Manager

Menu Screen

Map Selection Screen

Play Screen

Các Screen có thể chuyển đổi trực tiếp

Lớp Game quản lý việc thực hiện việc giữ cho game hoạt động và chuyển đổi giữa các Screen với nhau. Ví dụ, khi bắt đầu game, người dùng ở Menu Screen. Khi người dùng đang ở Menu Screen và muốn chuyển sang Map Selection Screen, người dùng bấm nút SinglePlayer. Lập tức, Menu Screen gửi một lệnh chuyển đổi Screen cho Game Manager và Game Manager thực hiện chuyển đổi sang Map Selection Screen.

2. Phân tích chương trình:

2.1. Các đối tượng chính cần biết của LibGDX:

2.1.1. Class Game:

Lớp Game cung cấp bởi LibGDX là một lớp ảo và được implement từ ApplicationListener. 3 hàm căn bản nhất chúng ta cần biết ở lớp này là:

* **create():** Hàm này được gọi 1 lần duy nhất ngay khi game vừa bắt đầu.
* **render():** Được gọi mỗi frame (Các frame thường được gọi sau mỗi 1/60s). Mọi hoạt động của game như vẽ các texture, sprite, handle input đều xảy ra ở hàm render.
* **dispose():** Giải phóng các tài nguyên cần thiết để tránh memory leak như texture, sound, …

Ngoài ra LibGDX còn cung cấp cho lớp Game này một số hàm hữu dụng khác như:

* **pause():** Được gọi khi game bị dừng. Thường là khi game không được kích hoạt hoặc là không được nhìn thấy trên màn hình.
* **resume():** Được gọi khi game được tiếp tục từ trạng thái đang pause() ở trên.
* **resize(int width, int height):** Được gọi khi game bị thay đổi kích thước màn hình.
* **setScreen(Screen screen):** Chỉ cho lớp Game biết Screen nào nên được render. Chi tiết hơn về hàm này sẽ được nói thêm ở các mục bên dưới.

Hàm **setScreen(Screen screen)** là một hàm rất quan trọng trong việc quản lý các Screen vì vậy chúng ta cần phải lưu ý tận dụng hàm này.

2.1.2. Interface Screen:

Chúng ta hoàn toàn có thể viết toàn bộ nội dung game vào class Game ở trên vì dường như nó đã cung cấp đủ các phương thức cần thiết để cho một game có thể hoạt động bình thường. Tuy nhiên, việc viết tất cả nội dung game vào chỉ một class Game rất khó để quản lý. Vì vậy, LibGDX cung cấp sẵn interface Screen để việc quản lý các cảnh game dễ dàng hơn.

Các hàm căn bản nhất của Interface Screen là:

* render(float delta): Tương tự như lớp GameManager, hàm này được gọi mỗi frame và mọi hoạt động của game được diễn ra tại đây.
* dispose(): Giải phóng các tài nguyên đã cấp.

2.2. Các đối tượng chính trong game Run:

2.2.1. Lớp GameManager:

**Extends class Game**

Được kế thừa từ lớp **Game** từ LibGDX framework, như đã nói ở trên, **GameManager** thực hiện việc quản lý các Screen và các Resource cần thiết.

3 hàm căn bản của lớp **GameManager** là:

* **create():** Hàm này được gọi 1 lần duy nhất ngay khi game vừa bắt đầu.
* **render():** Được gọi mỗi frame (Các frame thường được gọi sau mỗi 1/60s). Mọi hoạt động của game như vẽ các texture, sprite, handle input đều xảy ra ở hàm render.
* **dispose():** Giải phóng các tài nguyên cần thiết để tránh memory leak như texture, sound, …

Khi bắt đầu ứng dụng, lớp GameManager thực hiện việc tạo ra 1 **SpriteBatch** và các **Screen** cần thiết.

**SpriteBatch** là lớp phải được sử dụng để có thể thực hiện việc vẽ các vật thể lên màn hình, ví dụ như các texture, sprite, …

(Có thể đọc thêm về các lớp để thực hiện vẽ đối tượng tại đây: <https://github.com/libgdx/libgdx/wiki/Spritebatch%2C-Textureregions%2C-and-Sprites>)

Phải nói thêm, việc tạo ra 1 SpriteBatch ở mỗi lớp Screen để thực hiện vẽ các đối tượng là hoàn toàn có thể. Tuy nhiên, việc này gây ra việc tiêu tốn tài nguyên vì thực sự chúng ta chỉ cần dùng 1 SpriteBatch cho toàn bộ game. Vì vậy, tạo ra 1 SpriteBatch ở GameManager là hợp lý.

Tạo ra 1 SpriteBatch rất đơn giản, chúng ta chỉ cần gọi:

batch = new SpriteBatch();

Lệnh này nên được thực hiện ở hàm create() của lớp GameManager.

Các Screen căn bản cũng được tạo tại hàm create() của lớp này:

menuScreen = new MenuScreen(this);  
mapSelectionScreen = new MapSelectionScreen(this);

Các lớp screen này sẽ được giải thích rõ hơn ở các phần sau.

Bởi vì game sẽ bắt đầu với MenuScreen, sau khi khởi tạo các lớp và tài nguyên cần thiết thì tại hàm **create()**, ta có thể thực hiện:

setScreen(menuScreen);

**setScreen(Screen screen)** là một hàm được cung cấp sẵn bởi lớp Game cho phép GameManager biết được nên render Screen nào.

Ngoài ra, việc quản lý một lúc nhiều hình ảnh và âm thanh khá là rắc rối và hơn nữa, có một vài tài nguyên chúng ta có thể sử dụng lại cho nhiều Screen khác nhau, LibGDX cung cấp lớp đối tượng **AssetManager** cho phép ta quản lý các tài nguyên này một cách dễ dàng hơn.

Trong hàm **create()**:

assetManager = new AssetManager();

Và việc load các tài nguyên được thực hiện như sau:

assetManager.load(String fileName, Class<T> type);

fileName là đường dẫn đến tài nguyên và type là loại tài nguyên (Có thể là Texture.Class, Sound.Class, …)

2.1.2. Lớp MenuScreen

**Implements interface Screen**

Các hàm căn bản của lớp **MenuScreen** nói riêng và các lớp implement từ **Screen** nói chung là:

* constructor: Khởi tạo các lớp, các tài nguyên cần thiết.
* render(float delta): Tương tự như lớp GameManager, hàm này được gọi mỗi frame và mọi hoạt động của game được diễn ra tại đây.
* dispose(): Giải phóng các tài nguyên đã cấp.

Các hàm căn bản của lớp MenuScreen gần như không có gì quá khác biệt với các hàm căn bản của lớp GameManager. Điều này là dễ hiểu vì thực chất, lớp GameManager thực hiện việc render cho lớp MenuScreen ở hàm render().

@Override  
public void render () {  
 super.render();  
 ...

Một lỗi thường thấy ở việc viết hàm render() cho lớp GameManager là việc bỏ qua super.render(). Thực chất, hàm này thực hiện việc gọi hàm render(float delta) của lớp Screen hiện tại.

Interface Screen không cung cấp phương thức create() như là lớp Game, vì vậy chúng ta tạo ra constructor để khởi tạo các tài nguyên mà chúng ta mong muốn.

public MenuScreen(GameManager gameManager)

constructor của lớp MenuScreen nhận vào một đối số là **GameManager**. Như đã phân tích ở lớp Game, việc sử dụng lại **SpriteBatch** là dư thừa và chúng ta cũng cần phải lấy các tài nguyên cần thiết từ **AssetManager.** Vì vậy, chúng ta truyền vào đối số GameManager để giúp chúng ta lấy ra được những đối tượng này.

Lớp MenuScreen thực hiện việc render các button như SinglePlayer button, MultiPlayer button ChangeCharacter button. Tuy nhiên trước khi đi sâu vào việc

2.1.3. Lớp MapSelectionScreen

**Implements class Screen**

2.1.4. Lớp PlayScreen

**Implements class Screen**

3. Giao diện:

III.

IV. Kết luận:

1. Kết luận:

2. Hướng mở rộng:

Map editor in game