**ĐẠI HỌC QUỐC GIA TP. HCM**

ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

---------------🙦 🕮 🙤---------------

**BÁO CÁO ĐỒ ÁN NHẬP MÔN  
 PHÁT TRIỂN GAME**

**Ninja Gaiden (NES)**



Stage 3-1, 3-2, 3-3 (BOSS BASAQUER)

*Giảng viên hướng dẫn*

**ThS. Nguyễn Vĩnh Kha**

*Sinh viên thực hiện*

**Lê Quốc Phương**

*MSSV*

**17520134**

Mục lục

[Lời cảm ơn 2](#_Toc11145489)

[I. Framework 3](#_Toc11145490)

[1. Keyboard (SINGLETON): xử lý bàn phím 3](#_Toc11145491)

[2. Graphic (SINGLETON): xử lý giao diện 3](#_Toc11145492)

[3. Sprites: hình ảnh được xử lý 3](#_Toc11145493)

[4. Textures (SINGLETON): Kho hình ảnh 4](#_Toc11145494)

[5. Animation: Mô phỏng chuyển động 4](#_Toc11145495)

[6. gnhpSound (SINGLETON): Xử lý âm thanh 5](#_Toc11145496)

[7. Một số hàm để hỗ trợ 5](#_Toc11145497)

[8. Tổng kết một chút kinh nghiệm về Framework 6](#_Toc11145498)

[II. Kỹ thuật trong game 7](#_Toc11145499)

[1. State Pattern 7](#_Toc11145500)

[2. Spatial Partition (Grid) 7](#_Toc11145501)

[3. Object Pooling (Object pool pattern) 8](#_Toc11145502)

[4. Tag và Layer 8](#_Toc11145503)

[5. SweptAABB 8](#_Toc11145504)

[6. Singleton 9](#_Toc11145505)

[Lời kết 10](#_Toc11145506)

[Tự đánh giá 10](#_Toc11145507)

[Hướng phát triển 10](#_Toc11145508)

Lời cảm ơn

**Đầu tiên, em xin gởi lời cảm ơn chân thành đến tập thể quý Thầy Cô Trường Đại học Công nghệ thông tin – Đại học Quốc gia TP.HCM và quý Thầy Cô khoa Nhập môn Công Nghệ Phần Mềm đã giúp cho nhóm chúng em có những kiến thức cơ bản làm nền tảng để thực hiện đề tài này.**

**Đặc biệt, em xin gửi lời cảm ơn và lòng biết ơn sâu sắc nhất tới Thầy Nguyễn Vĩnh Kha. Đã giúp em hoàn thành tốt báo cáo môn học của mình.**

**Trong thời gian một học kỳ thực hiện đề tài, em đã vận dụng những kiến thức nền tảng đã tích lũy đồng thời kết hợp với việc học hỏi và nghiên cứu những kiến thức mới. Từ đó, em vận dụng tối đa những gì đã thu thập được để hoàn thành một báo cáo đồ án tốt nhất. Tuy nhiên, trong quá trình thực hiện, em không tránh khỏi những thiếu sót. Chính vì vậy, em rất mong nhận được những sự góp ý từ phía các Thầy Cô nhằm hoàn thiện những kiến thức mà nhóm chúng em đã học tập và là hành trang để nhóm chúng em thực hiện tiếp các đề tài khác trong tương lai.**

**Xin chân thành cảm ơn các quý Thầy Cô !**

1. **Framework**
2. Keyboard (SINGLETON): xử lý bàn phím

* Sử dụng DirectInput8
* 3 hàm cơ bản:
  + GetKeyDown(KeyCode);
  + GetKeyUp(KeyCode);
  + GetKey(KeyCode);
* Trong đó GetKeyDown xử lý sự kiện phím nhấn xuống trong frame đó, GetKeyUp xử lý sự kiện thả phím trong frame đó. GetKey xử lý trạng thái của phím

1. Graphic (SINGLETON): xử lý giao diện

* Sử dụng Direct3DDevice
* Graphic hỗ trợ vẽ chữ để tạo bảng điểm và trạng thái trong game bằng hỗ trợ của DirectX là ID3DXFont

void DrawString();

void DrawCustomZone(RECT r, std::string content);

* Graphic còn hỗ trợ vẽ texture lên tại một vị trí cố định trên màn hình (Screen coordinate)

void Draw(float x, float y, LPDIRECT3DTEXTURE9 texture);

1. Sprites: hình ảnh được xử lý

* Sử dụng LPDIRECT3DTEXTURE9 để lưu trữ Texture kết hợp với một RECT (hoặc BoxCollider là bản float của RECT) để cắt thành sprite nhỏ
* Hỗ trợ vẽ ngay trong sprite bằng hàm

void Draw(D3DXVECTOR3 position = D3DXVECTOR3(), BoxCollider r = BoxCollider(), D3DXCOLOR colorKey = D3DCOLOR\_XRGB(255, 255, 255), bool reverse = false);

* Trong đó:
* *position: vị trí*
* *r: hình chữ nhật để crop hình trong texture (thường không dùng và được khởi tạo ngay khi tạo sprite)*
* *colorKey: pha màu*
* *reverse: đổi chiều*

1. Textures (SINGLETON): Kho hình ảnh

* Kho lưu trữ Textures để tái sử dụng các texture, tránh trường hợp một texture được tải lên nhiều lần
* Hoạt động: gồm 2 trạng thái thêm vào và lấy ra

void Add(int id, LPCSTR path, D3DCOLOR transparentColor);

LPDIRECT3DTEXTURE9 Get(unsigned int i);

1. Animation: Mô phỏng chuyển động

* Hoạt động mô phỏng đơn giản theo animation của Engine game Unity3D
* Có thêm các AnimationFrame (KeyFrame trong Unity3D). Vì các chuyển động theo vị trí đều dựa trên các sự kiện trong game, còn animation sẽ giữ nguyên vị trí nên AnimationFrame chỉ có mỗi sprite, nên sẽ cho AnimationFrame kế thừa sprite và có thêm một thuộc tính để thực hiện chuyển động



DWORD time;

* Animation có thêm một số hàm hỗ trợ khác:

*Render hoạt động như hàm Draw của sprite*

void Render(D3DXVECTOR3 position = D3DXVECTOR3(), BoxCollider sourceRect = BoxCollider(), D3DXCOLOR colorKey = D3DCOLOR\_XRGB(255, 255, 255), bool isReverse = false);

void AddFrames(LPDIRECT3DTEXTURE9 texture, int rows, int columns, float timePerFrame = 0.1f, D3DCOLOR colorKey = D3DCOLOR\_XRGB(255, 255, 255));

void AddFramesA(LPDIRECT3DTEXTURE9 texture, int stRows, int fnRows, int columns, int rowNumber, int columnNumber, float timePerFrame = 0.1f, D3DCOLOR colorKey = D3DCOLOR\_XRGB(255, 255, 255));

AddFrames: đưa vào rows và columns của 1 chuyển động duy nhất (texture chỉ có 1 chuyển động duy nhất)

AddFramesA: đưa vào startRows, finishRows của chuyển động cần lấy, thêm số lượng columns muốn lấy, và rowNumber, columNumber là số lượng hàng và cột của cả project

* Hai hàm này chỉ làm theo để phù hợp với logic của người code

*Kiểm tra Frame cuối*

bool IsLastFrame(double dt);

*Lấy % thực hiện animation (0 -> 1 = 0% -> 100%)*

double GetPercentTime();

1. gnhpSound (SINGLETON): Xử lý âm thanh

Sử dụng:

LPDIRECTSOUND8

LPDIRECTSOUNDBUFFER

WAVE file

*Cách sử dụng: trong giáo trình Nhập môn Phát triển Game của trường Đại học Công nghệ Thông tin – Đại học Quốc gia TP.HCM*

1. Một số hàm để hỗ trợ

Debug.h Debug.cpp

* Thực ra trong debugapi có hỗ trợ hàm giúp debug như OutputDebugString, nhưng debug viết ra theo dạng custom để dễ theo dõi các biến trong từng frame hơn

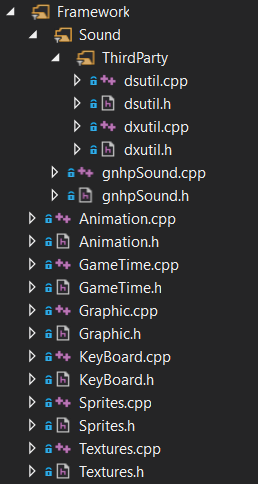
*Nguồn tham khảo:*

<https://github.com/dungdna2000/gamedev-intro-tutorials>

* MyHelper: Có sử dụng một số kỹ thuật như Clamp, Lerp (linear interpolation), Distance… Đều mô phỏng từ Engine Game Unity3D

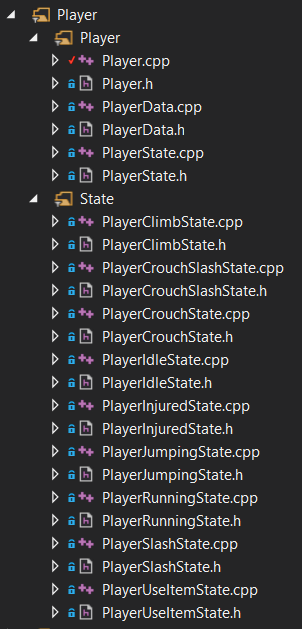
1. Tổng kết một chút kinh nghiệm về Framework

* Framework thực sự directX đã hỗ trợ rất nhiều, người lập trình không phải làm gì nhiều ngoài việc khởi tạo và xây dựng hàm hỗ trợ
* Việc quan trọng là tương tác giữa các nội dung của framework



1. Kỹ thuật trong game
2. State Pattern

* Chia nhỏ các hoạt động của nhân vật, vật thể (trong game là Entity) ra thành các state
* Mục đích: dễ dàng quản lý, xử lý cho mỗi trạng thái.
* Ví dụ dưới cho player, tương tự enemy hoặc weapon có trạng thái phức tạp cũng sẽ sử dụng pattern này



* *Nguồn tham khảo:*

<https://loctho95.blogspot.com/2017/01/chuong-ix-state-machine-player-bot.html>

1. Spatial Partition (Grid)

* Chia nhỏ world (game scene) thành từng khối nhỏ, cũng có thể xem là matrix và đặt các object vào trong đó theo vị trí, mỗi ô gọi là **Unit**
* Mục đích cơ bản: tránh việc so sánh collide quá nhiều object trong một frames cũng giống như Quadtree, tất nhiên việc sử dụng grid sẽ đơn giản hơn Quadtree
* *Nguồn tham khảo:*

<http://gameprogrammingpatterns.com/spatial-partition.html>

1. Object Pooling (Object pool pattern)

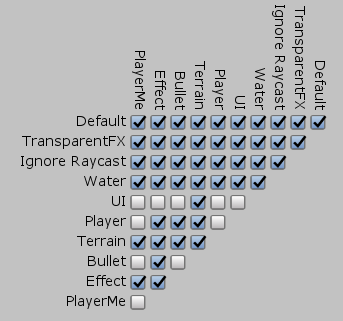
* Đặt các vật thể vào một “pool”, nếu muốn sử dụng chỉ cần lấy ra, không sử dụng thì ném vào lại “pool”.
* Mục đích: tái sử dụng
* *Nguồn tham khảo:*

<https://learn.unity.com/tutorial/object-pooling>

1. Tag và Layer

* Mỗi vật thể trong thế giới game đều có một tag riêng và một layer chúng thuộc về
* Trong DirectX chúng ta sẽ đơn giản hóa việc sử dụng Tag và Layer so với Engine Game Unity3D
* Mục đích: Tag: xử lý trường hợp đặc biệt, Type: xét va chạm

Ví dụ:

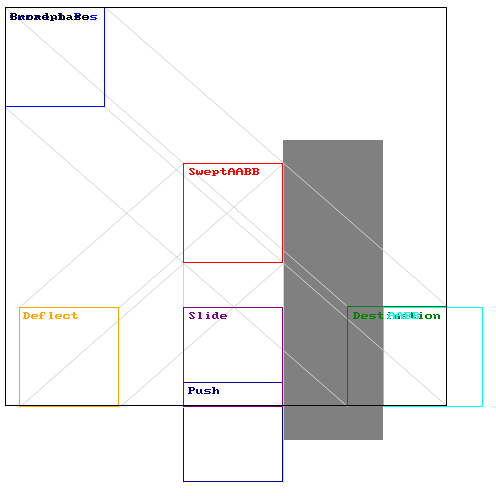


*Nguồn tham khảo:*

[https://docs.unity3d.com/Manual//class-TagManager.html](https://docs.unity3d.com/Manual/class-TagManager.html)

1. SweptAABB

* Xử lý va chạm chính xác
* Mục đích: xét trong frame đó có khả năng va chạm không, tránh việc xử lý va chạm theo truyền thống xuất hiện case lỗi

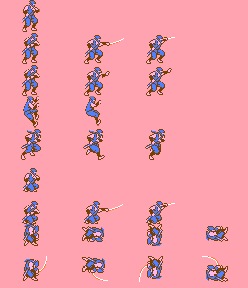
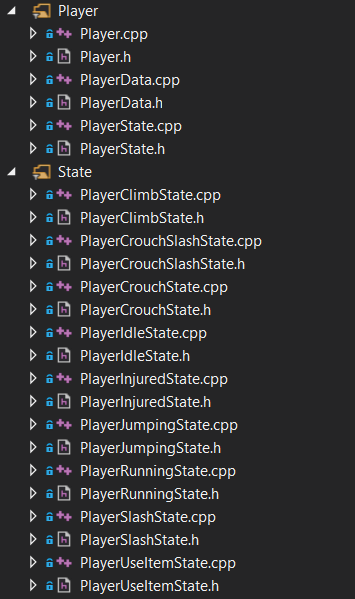


*Nguồn hình: giáo trình Nhập môn Phát triển Game – Đại học Công nghệ Thông tin – Đại học Quốc gia TP.HCM*

1. Singleton

* Singleton: mỗi class chỉ có duy nhất một đối tượng được tạo ra
* Mục đích: dễ dàng sử dụng, quản lý
* *Trong game directX của mình mỗi scene sẽ có một singleton riêng, nghĩa là singleton sẽ không được lưu lại sau khi chuyển scene khác với DontDestroyOnLoad của Unity3D*

1. Hệ thống game
   * + 1. Nhân vật

* Ryu: nhân vật điều khiển, gồm 9 state theo thứ tự: đứng yên, chém, sử dụng item, leo trèo, chạy, ngồi, ngồi chém, nhảy, nhảy chém

Hệ thống state trong game sử dụng state pattern để xử lý qua lại giữa các state. Chi tiết xem ở code

* + - 1. Quái vật, địch (Enemy)

|  |  |  |  |
| --- | --- | --- | --- |
| ENEMY | INFORMATION | AI | Vũ khí |
| C:\Users\ThinkKING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\enemy11.pngSparta | DAMAGE: 1  Points: 100 | Đi và tấn công bình thường, hết terrain tự động quay lại |  |
| Cat | DAMAGE: 1  Points: 200 | Di chuyển bằng cách nhảy, nếu nhảy đúng đoạn hết terrain thì quay lại |  |
| Thrower | DAMAGE: 1  Points: 300 | Di chuyển qua lại, ném knife về phía player. Thời gian ném, di chuyển là không cố định | DAMAGE: 1  C:\Users\ThinkKING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\knife.png |
| Eagle | DAMAGE: 3  Points: 200 | Bay trên không, xà xuống player với gia tốc x lớn, y nhỏ |  |
| Soldier  C:\Users\ThinkKING\source\repos\LearningDirectX2\LearningDirectX2\Resources\Sprites\soldier1.png | DAMAGE: 1  HP: 1  Points: 200 | Sau gần 2 state di chuyển, sẽ chuyển sang bắn 3 viên đạn hướng player, hết terrain tự động quay lại | DAMAGE: 1  C:\Users\ThinkKING\AppData\Local\Microsoft\Windows\INetCache\Content.Word\bulletspritesheet.png |
| Gunner | DAMAGE: 1  Points: 200 | Không di chuyển, sau 1s bắn 1 viên đạn lớn hướng player | DAMAGE:2 |
| Runner | DAMAGE: 1  Points: 300 | Di chuyển bằng cách chạy, hết terrain tự động nhảy |  |
| Bat | DAMAGE: 1  Points: 100 | Di chuyển lên xuống hướng player |  |
| Basaquer  C:\Users\ThinkKING\source\repos\LearningDirectX2\LearningDirectX2\Resources\Sprites\boss1.png | DAMAGE: 2  HP: 16  Points: custom | Nhảy qua lại giữa 2 border của stage, sau khi nhảy sẽ bắn 3 darts cố định theo số lần nhảy tỷ lệ 3 : 3 : 4 | DAMAGE:2 |

* + - 1. Hệ thống Item (Trong 3 stage 3-1, 3-2, 3-3)

|  |  |  |
| --- | --- | --- |
| ITEM | INFORMATION | TẤN CÔNG |
| SpiritPoints 5  Ninja Gaiden NES item ability blue.png | SPIRIT POINTS + 5 |  |
| Spirit Points 10  Ninja Gaiden NES item ability red.png | SPIRIT POINTS + 10 |  |
| Scores 500  Ninja Gaiden NES item bonus blue.png | Scores + 500 |  |
| Scores 1000  Ninja Gaiden NES item bonus red.png | Scores + 1000 |  |
| Time Freeze  Ninja Gaiden NES item time.png | Ngưng động thời gian trong 5 giâY (LOGIC: chỉ ENEMY) |  |
| Health  Ninja Gaiden NES item health.png | Health + 6 |  |
| Throwing Star  Ninja Gaiden NES item throwing star.png | Item: Shuriken | Di chuyển theo hướng player ném |
| Windmill Star  Ninja Gaiden NES item boomerang star.png | Item:  Big Shuriken | Di chuyển theo hướng player ném ra và quay trở lại như boomerang, nếu đụng player sẽ thu hồi |
| Flames  Ninja Gaiden NES item flame.png | Item:  Flames | Phóng ra 3 tập ngọn lửa, mỗi tập di chuyển theo một góc khác nhau |

1. Vận hành Game
   * + 1. Giai đoạn khởi tạo

Lời kết

## Tự đánh giá

Đánh giá mức độ hoàn thiện Game:

* Hoàn thành khoảng 95% game gốc (tính riêng 3 stage 3-1, 3-2, 3-3 (BOSS))

Một số kiến thức quan trọng được học và áp dụng:

* Grid
* Boundary collision detection
* Viewport
* Camera
* Matrix
* Sprite
* World map
* Tile map
* …

Trình bày báo cáo:

* Vì giới hạn không quá 10 trang nên **có thể** còn nhiều xử lý chưa được trình bày
* Các phần xử lý quan trọng sẽ được ưu tiên trong bài báo cáo ạ

## Hướng phát triển

* Vì game đã chia ra rất nhiều scene và có cơ chế tự xây dựng destructor ổn, nên có thể phát triển thêm nhiều scene hơn, nhiều nhân vật hơn… mà không (hoặc ít) bị đầy bộ nhớ