ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**



**Họ và tên sinh viên nhóm tác giả đồ án:**

|  |  |  |
| --- | --- | --- |
| Sinh viên thực hiện: | | |
| STT | Họ tên | MSSV |
| 1 | Lê Huỳnh Thăng | 17520155 |
| 2 | Dương Thành Đạt | 17520040 |

**NINJA GAIDEN**

**Nhóm SE1 – Lớp SE102.J21**

Đồ án môn Nhập môn Phát triển game

GIÁO VIÊN HƯỚNG DẪN: Nguyễn Vĩnh Kha

Mục lục

[**Giới thiệu** 3](#_Toc11439368)

[**Các lớp chính** 4](#_Toc11439369)

[**1.** **DxFramework:** 4](#_Toc11439370)

[*a)* *dxgraphics:* 4](#_Toc11439371)

[*b)* *dxinput:* 4](#_Toc11439372)

[*c)* *dxsound:* 4](#_Toc11439373)

[2. **Original**: 4](#_Toc11439374)

[*a)* *Camera:* 4](#_Toc11439375)

[*b)* *Grid:* 5](#_Toc11439376)

[*c)* *Sprite:* 5](#_Toc11439377)

[*d)* *Tiles:* 5](#_Toc11439378)

[*e)* *State:* 5](#_Toc11439379)

[*f)* *ScoreBoard:* 6](#_Toc11439380)

[*g)* *Stage:* 6](#_Toc11439381)

[**3.** **SupportFunction:** 6](#_Toc11439382)

[*a)* *Collision:* 6](#_Toc11439383)

[**4.** **Object:** 6](#_Toc11439384)

[*a)* *Enemies:* 7](#_Toc11439385)

[**5.** **Game:** 8](#_Toc11439386)

[**6.** **winmain:** 8](#_Toc11439387)

[**Tổng kết** 9](#_Toc11439388)

[**Tài liệu tham khảo** 10](#_Toc11439389)

# **Giới thiệu**

Với sự hướng dẫn của thầy Nguyễn Vĩnh Kha, nhóm em đã sử dụng Visual Studio 2017 và Window SDK 10 cùng thư viện chính là DirectX9 SDK – June 2010 để hoàn thành đồ án mô phỏng game ninja gaiden. Bên cạnh đó, nhóm đã vận dụng kiến thức lập trình hướng đối tượng và sử dụng ngôn ngữ C++ để giải quyết các bài toán phát sinh khi phát triển game như va chạm, phân hoạch không gian, quản lí trạng thái của nhân vật,...

Trong báo cáo này sẽ mô tả tổng quan các lớp quan trọng, chức năng và cách vận hành của chúng trong game. Đồng thời, tự đánh giá một số ưu và khuyết điểm, cũng như một số khó khăn khi thực hiện đồ án

# **Các lớp chính**

1. **DxFramework:**

Đây là framework để xử lí các vấn đề liên quan chủ yếu đến thư viện DirectX như xử lí âm thanh, hình ảnh và input

1. *dxgraphics:*

Xử lí các vấn đề về hình ảnh:

* int Init\_Direct3D: khởi tạo các biến cần dùng cho việc tải hình ảnh (vd: LPDIRECT3D9, LPDIRECT3DDEVICE9, LPDIRECT3DSURFACE9, LPD3DXSPRITE)
* LPDIRECT3DSURFACE9 LoadSurface: tải hình ảnh lên trên surface
* LPDIRECT3DTEXTURE9 LoadTexture: tải hình ảnh lên để vẽ trên spritehandler

1. *dxinput:*

Xử lí các vấn đề về input:

* int Init\_DirectInput: khởi tạo các biến cần dùng cho việc xử lí input (vd: LPDIRECTINPUT8, LPDIRECTINPUTDEVICE8, LPDIRECTINPUTDEVICE8)
* int Init\_Keyboard: khởi tạo các định dạng cho bàn phím
* void Poll\_Keyboard: lấy trạng thái của bàn phím
* int Key\_Down: trả về xem phím cần kiểm tra có được nhấn
* int Key\_Up: trả về xem phím cần kiểm tra có được thả
* void Kill\_Keyboard: xóa và trả quyền sử dụng bàn phím cho hệ thống
* void Poll\_Mouse: lấy trạng thái của chuột
* int Init\_Mouse: khởi tạo các định dạng cho chuột
* int Mouse\_Button : trả về xem phím chuột cần kiểm tra có được nhấn
* int Mouse\_X: lấy vị trí X của chuột
* int Mouse\_Y: lấy vị trí Y của chuột
* void Kill\_Mouse: xóa và trả quyền sử dụng chuột cho hệ thống

1. *dxsound:*

Cùng với hai thư viện có sẵn là DSUtil và dxutil xử lí các vấn đề âm thanh

* int init\_SoundManager: khởi tạo các biến cần dùng cho việc xử lí âm thanh (vd: CsoundManager)
* int addSound: thêm âm thanh
* void playSound: chạy âm thanh
* void stopSound: dừng âm thanh

1. **Original**:

Là những thư viện gốc – gồm những đối tượng cơ sở quan trọng để phát triển game nói chung

1. *Camera:*

Là một đối tượng đi theo nhân vật để “quay lại” những cảnh vật xung quanh nó

* D3DXVECTOR3 WorldToView: chuyển tọa độ trong thế giới thực thành tọa độ trong game
* void Update: cập nhật lại vị trí của camera

1. *Grid:*

Dùng để phân hoạch không gian, giúp giảm chi phí tính toán trong quá trình xử lí. Ở đây, Grid chỉ mới áp dụng cho phân hoạch tường và đất (Wall)

* void GridClear: xóa Grid
* void Add: thêm một Wall vào trong Grid
* vector<Wall\*> WallOut: lấy ra các Wall nằm trong vùng cần xét xử lí

1. *Sprite:*

Dùng để lưu thông tin của hình cần vẽ và vẽ nó lên spritehandler. Dùng cho các Sprite có kích thước giống nhau

* void Render(Camera camera): vẽ texture lên spritehandler với tâm là trọng tâm của hình
* void Render(Camera camera, RECT renderRect): vẽ texture lên spritehandler với tâm là ở giữa phía dưới chân hình ứng với hình chữ nhật bao khuôn đã cho
* void Render(Camera camera, RECT renderRect, D3DXVECTOR3 center): vẽ texture lên spritehandler với tâm đã cho ứng với hình chữ nhật bao khuôn đã cho
* virtual void KillSprite: xóa các dữ liệu cần thiết trước khi hủy Sprite
* Box ToBox: đóng bao Sprite thành hình chữ nhật, thường dùng để xét va chạm
* virtual void RectUpdate: chuyển sang ô Sprite khác

1. *Tiles:*

Sprite của nhân vật không phải lúc này cũng cùng một kích cỡ. Dùng Tiles kèm với phương thức Render có truyền tham số hình chữ nhật trong Sprite sẽ khắc phục vấn đề này. Tiles có sử dụng thư viện hỗ trợ đọc file .xml sẽ được giới thiệu ở mục SupportFunction

* RECT getRectLocation: lấy hình chữ nhật tại vị trí sprite thứ ID
* void TilesClear: xóa các dữ liệu cần thiết trước khi xóa Tiles

1. *State:*

Dùng để quản lí các trạng thái của nhân vật, thường là nhân vật chính. Bằng cách kế thừa và đa hình, việc quản lí các trạng thái này sẽ dễ hơn. Ở đây chỉ giới thiệu lớp cha là lớp State:

* virtual int HandleInput: xử lí input và xử lí các ngoại lệ để chuyển trạng thái
* virtual void UpdateVelocity: cập nhật vận tốc của nhân vật theo trạng thái
* virtual void UpdateStatus: cập nhật vị trí của nhân vật theo trạng thái
* virtual void UpdateCollision: cập nhật va chạm của nhân vật theo trạng thái
* virtual Box ToBoxSword: đóng bao vũ khí nhân vật thành hình chữ nhật, thường dùng để xét va chạm
* virtual Box ToBox: đóng bao nhân vật thành hình chữ nhật, thường dùng để xét va chạm

1. *ScoreBoard:*

Bảng điểm và những thông tin cần thiết khi chơi game

* void Update: cập nhật ScoreBoard
* void Render: vẽ ScoreBoard
* void ScoreBoardDelete: xóa các dữ liệu cần thiết trước khi xóa ScoreBoard

1. *Stage:*

Dùng để lưu các dữ liệu liên quan đến màn chơi (vd: đường dẫn đến các file source, vị trí bắt đầu của nhân vật chính)

1. **SupportFunction:**

Bao gồm những thư viện hỗ trợ cho các công xử lí liên quan. Các thư viện này chỉ gồm các phương thức và không chia class

1. *Collision:*

Dùng để kiểm tra va chạm và chồng lấn hình

* float SweptAABB: kiểm tra va chạm của 2 hình chữ nhật và trả về thời gian va chạm và hướng va chạm
* bool OverlappedBox: kiểm tra 2 hình chữ nhật có chồng lấn lên nhau không

1. *TinyXml:*

Đây là một thư viện hỗ trợ tìm thấy trên mạng để hỗ trợ Tiles đọc các file .xml. Cùng với các phần mềm cắt và tạo SpriteSheet, đây là những bộ đôi khá mạnh trong việc hỗ trợ vẽ Sprite. Những phần mềm này sẽ được giới thiệu ở mục tài liệu tham khảo.

1. *debug:*

Thư viện này hỗ trợ trong việc debug. Khác với các chương trình console bình thường, việc debug trong game gặp khá nhiều khó khăn vì việc xử lí đầu vào sẽ khiến game chạy trong chế độ debug (có breakpoint) khác với việc chạy game bình thường. Chính vì vậy, việc viết những thông tin cần quan sát ra Output (debug của chương trình) sẽ khiến việc sửa lỗi dễ hơn.

* void DebugOut: sử dụng như printf, để viết ra Output những thông tin cần quan sát

1. **Object:**

Bao gồm những vật thể xuất hiện trong game, cụ thể là game Ninja Gaiden

1. *Enemies:*

Gồm những thuộc tính và phương thức cần thiết để tạo đối thủ trong game. Đây là lớp cha và sẽ được thừa kế bởi các quái vật cụ thể như chim, xạ thủ,... Áp dụng kế thừa và đa hình để xây dựng các lớp này. Ở đây chỉ giới thiệu lớp cha là Enemy

* virtual void UpdateVelocity: cập nhật vận tốc của quái vật
* virtual void UpdatePosition: cập nhật vị trí của quái vật
* virtual void UpdateFrame: cập nhật khung bao để vẽ Sprite
* virtual LPEnemy Update: gọi các hàm trên để cập nhật quái vật và xử lí thêm nếu có. Có thể trả về “quái con” như đạn
* virtual void Render: vẽ quái vật lên Spritehandler
* virtual Box ToBox: đóng bao quái vật để xét va chạm
* virtual void EnemyDelete: xử lí trước khi xóa quái vật
* virtual void DiedUpdate: cập nhật khi quái vật vào trạng thái bị hạ gục, thường là cập nhật về hoạt ảnh
* virtual void DiedRender: vẽ quái vật khi nó trong trạng thái bị hạ gục

1. *Item:*

Lớp vật phẩm dùng để hỗ trợ cho nhân vật chính

* void Update: cập nhật lại Item như vị trí, đã được ăn hay chưa,...
* void ItemDelete: xử lí trước khi xóa Item
* void Render: vẽ Item lên Spritehandler
* Box ToBox: đóng báo Item để xét va chạm

1. *Map:*

Bản đồ hay còn gọi là nền của game

* void KillSprite: xử lí trước khi xóa Map
* void Render: vẽ bản đồ lên Spritehandler

1. *Ryu:*

Là nhân vật chính của game

* void ChangeState: kiểm tra và xử lí việc chuyển đổi State
* void UpdateState: cập nhật lại nhân vật chính dựa trên State hiện tại
* void UpdateCollision: kiểm tra va chạm với tường và xử lí
* void Update: gọi các hàm trên để cập nhật Ryu và xử lí thêm nếu có
* void Render: vẽ Ryu lên Spritehandler
* void ChangeStage: xử lí chuyển màn chơi đối với Ryu
* Box ToBox: đóng bao nhân vật chính để kiểm tra va chạm
* void Remove: xử lí trước khi xóa Ryu
* Box ToBoxSword: đóng bao kiếm của nhân vật Ryu nếu có
* Box ToSkillBox: đóng bao Skill phóng ra của Ryu nếu có
* void KillSkill: xử lí trước khi xóa Skill đã được phóng ra

1. *Skill:*

Là kĩ năng của nhân vật khi ăn vật phẩm kỹ năng. Áp dụng kĩ thuật kế thừa đa hình. Ở đây, chỉ giới thiệu lớp cha là Skill

* virtual void UpdateVelocity(LPSprite RyuSprite): cập nhật vận tốc của Skill
* virtual void Update(LPSprite RyuSprite): gọi hàm trên và xử lí các cập nhật khác
* virtual void SkillDelete(): xử lí trước khi xóa Skill
* virtual void Render(Camera camera): vẽ Skill lên Spritehandler
* virtual Box ToBox(): đóng bao Skill để xử lí va chạm

1. *Container:*

Là đồ chứa các vật phẩm

* LPItem Update: cập nhật trạng thái và khung vẽ của Container, nếu bị đánh sẽ làm rớt vật phẩm
* void Render: vẽ lên Spritehandler
* void ContainerDelete: xử lí trước khi xóa Container
* Box ToBox: đóng bao để xét xử lí va chạm

1. *Wall:*

Là đất hay cầu thang để nhân vật chính đứng

* Box ToBox: đóng bao để kiểm tra va chạm

1. **Game:**

Là môi trường giao tiếp giữa các vật thể trong game

* int Game\_Init: khởi tạo các giá trị cần thiết cho game
* void Game\_Run: chạy game, xử lí các vấn đề giao tiếp giữa các vật thể trong game và vẽ lên chúng
* void Game\_End: xử lí khi kết thúc game
* void Change\_Stage: xử lí việc chuyển màn chơi
* void Update(): cập nhật vật thể, xử lí va chạm giữa 2 vật thể khác loại, xử lí hiệu ứng của vật phẩm khi được ăn

1. **winmain:**

Là nơi khởi tạo cửa sổ và vòng lặp game, xử lí trạng thái pause game

# **Tổng kết**

Sau một thời gian làm đồ án, nhóm đã tìm hiểu và vượt qua một số khó khăn như xây dựng framework DirectX, áp dụng các kĩ thuật mới như phân hoạch không gian và state pattern, quản lí source code và làm việc theo nhóm,... Kết quả của đồ án lần này, nhóm cảm thấy front-end có thể đáp ứng được 75%-85% so với bản gốc. Tuy nhiên, nhóm cũng thừa nhận những khuyết điểm của đồ án lần này: source code có nhiều chỗ chưa gọn gàng; chỉ mới áp dụng phân hoạch không gian cho Wall, còn lại chỉ giảm bớt số lượng vật thể cần xử lí bằng cách xét tọa độ trong vùng Camera; gặp rắc rối khi dùng github;...

Cuối cùng, nhóm SE1 chúng em xin cảm ơn sự hướng dẫn của thấy Nguyễn Vĩnh Kha và Khoa Công nghệ phần mềm. Nhờ có thầy mà nhóm có thể hoàn thành đồ án Game lần này

# **Tài liệu tham khảo**

* Tài liệu chính: <https://drive.google.com/file/d/1OLmspsOMVon9o5f_wSlsId-BOuqm-Ze_/view?fbclid=IwAR3oJvBc75ApZE1EQLjzfyEjr8XRc0fuDCEAbH7K9GZYuqgvF8CANeAQAdc>
* Tạo và đọc Sprite Sheet: <https://www.stdio.vn/articles/sprite-animation-trong-directx-9-602>
* State Pattern: <https://www.stdio.vn/articles/design-pattern-state-pattern-261>
* Phân hoạch không gian: <http://gameprogrammingpatterns.com/spatial-partition.html>