

**ĐẠI HỌC QUỐC GIA TP HỒ CHÍ MINH**

**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN**

**KHOA CÔNG NGHỆ PHẦN MỀM**

**LỚP KTPM 2013**

**----------oOo----------**

****

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

**Đề tài: XÂY DỰNG GAME TƯƠNG TỰ SUPER MARIO BROS 3**

**Giảng viên hướng dẫn:**

**ThS. ĐINH NGUYỄN ANH DŨNG**

**Sinh viên thực hiện:**

**TRẦN NGỌC TÚ - 13520969**

**HUỲNH NGỌC THẮNG - 13520775**

**TRẦN HUY THỊNH - 13520830**

**LÊ HOÀNG SINH - 13520703**

**Thành phố Hồ Chí Minh, ngày 05 tháng 01 năm 2015**

MỤC LỤC

[1 Giới thiệu về game Super Mario Bros 3 1](#_Toc439721802)

[2 Giới thiệu đồ án 1](#_Toc439721803)

[3 Phân tích và thiết kế 1](#_Toc439721804)

[3.1 Phân tích 1](#_Toc439721805)

[3.1.1 Danh sách các class sử dụng trong game 1](#_Toc439721806)

[3.1.2 Sơ đồ các class kế thừa trong game 3](#_Toc439721807)

[3.2 Thiết kế: 4](#_Toc439721808)

[3.2.1 Thiết kế chi tiết các class quản lý đối tượng trong game 4](#_Toc439721809)

[3.2.2 Thiết kế map 18](#_Toc439721810)

[4. Các kiến thức áp dụng 19](#_Toc439721811)

[4.1. Phương pháp lập trình lập hướng đối tượng 19](#_Toc439721812)

[4.1.1. Khái niệm 19](#_Toc439721813)

[4.1.2. Đặc điểm của lập trình hướng đối tượng 20](#_Toc439721814)

[4.1.3. Trừu tượng hóa dữ liệu 20](#_Toc439721815)

[4.1.4. Tính đa hình 20](#_Toc439721816)

[4.1.5. Tính kế thừa 20](#_Toc439721817)

[4.1.6. Tính đóng gói dữ liệu 20](#_Toc439721818)

[4.1.7. Những ưu điểm của lập trình hướng đối tượng. 21](#_Toc439721819)

[4.1.8. Các thành phần trong lập trình hướng đối tượng 21](#_Toc439721820)

[4.2. Hướng đối tượng trong game super mario bros 3 21](#_Toc439721821)

[4.3. Kỹ thuật phân chia không gian bằng QuadTree 22](#_Toc439721822)

[4.3.1. Giới thiệu về cây tứ phân (QuadTree) 22](#_Toc439721823)

[4.3.2. Sử dụng Quadtree để phân hoạch không gian trong game 23](#_Toc439721824)

[*5.* Giao diện và cách chơi 24](#_Toc439721825)

[*5.1.* Giao diện game 24](#_Toc439721826)

[*5.2.* Cách chơi 25](#_Toc439721827)

[6. Nhận xét, kết luận và hướng phát triển 25](#_Toc439721828)

[6.1. Nhận xét 25](#_Toc439721829)

[6.2. Kết luận 25](#_Toc439721830)

[6.3. Hướng phát triển 25](#_Toc439721831)

# Giới thiệu về game Super Mario Bros 3

Game Super Mario Bros 3 là một tựa game nổi tiếng một thời, đã từng làm say mê hàng triệu bạn trẻ những năm đầu thế kỷ XXI nhưng cho đến tận bây giờ, độ phổ biến của nó vẫn không hề thay đổi. Trong phiên bản Super Mario Bros 3 này, nội dung được tập trung vào mario và anh trai của cậu ta là Luigi. Hai người sẽ dấn thân vào một nhiệm vụ vô cùng khó khăn là giải cứu cô công chúa Peach vô cùng xinh đẹp và những vị vua của 7 vương quốc khác nhau từ kẻ thù chính của họ là Bowser và những đứa con của hắn Koopaling. Cả hai người sẽ phải vượt qua tất cả 8 miền đất khác nhau để khôi phục lại vương quốc nấm. Super mario bros 3 được xây dựng theo cách chơi của phiên bản mario trước. Game được giới thiệu những tính năng mới được sử dụng để làm gia tăng khả năng của nhân vật.

Việc chơi game cũng hết sức đơn giản, bạn chỉ việc điều khiển cho nhân vật đi, chạy để ăn thật nhiều đồng xu. Tuy nhiên, đôi lúc, game vẫn đòi hỏi sự suy luận từ phía người chơi, bởi trong đó vẫn có những câu đố nhỏ đan xen vào game để tăng tích hấp dẫn cho dòng game mini này.

# Giới thiệu đồ án

Phiên bản gốc của game có rất nhiều màn chơi rất hấp dẫn, tuy nhiên trong phạm vi đồ án môn học này chỉ thực hiện cảnh 1 của game. Yêu cầu chương trình của chúng ta ở đây là làm sao thiết kế được 1 chương trình game giống (hoặc gần giống) như cảnh 1 trong game Super Mario Bros 3 trên trên nền tảng DirectX, dựa vào các phương pháp lập trình đã học. Cụ thể là sử dụng phương pháp lập trình hướng đối tượng kết hợp với các kĩ thuật khác như phân chia không gian sử dụng cây tứ phân (Quadtree), các kĩ thuật chuyển đổi tọa độ và sử dụng các thư viện đồ họa, thư viện âm thanh trong DirectX.

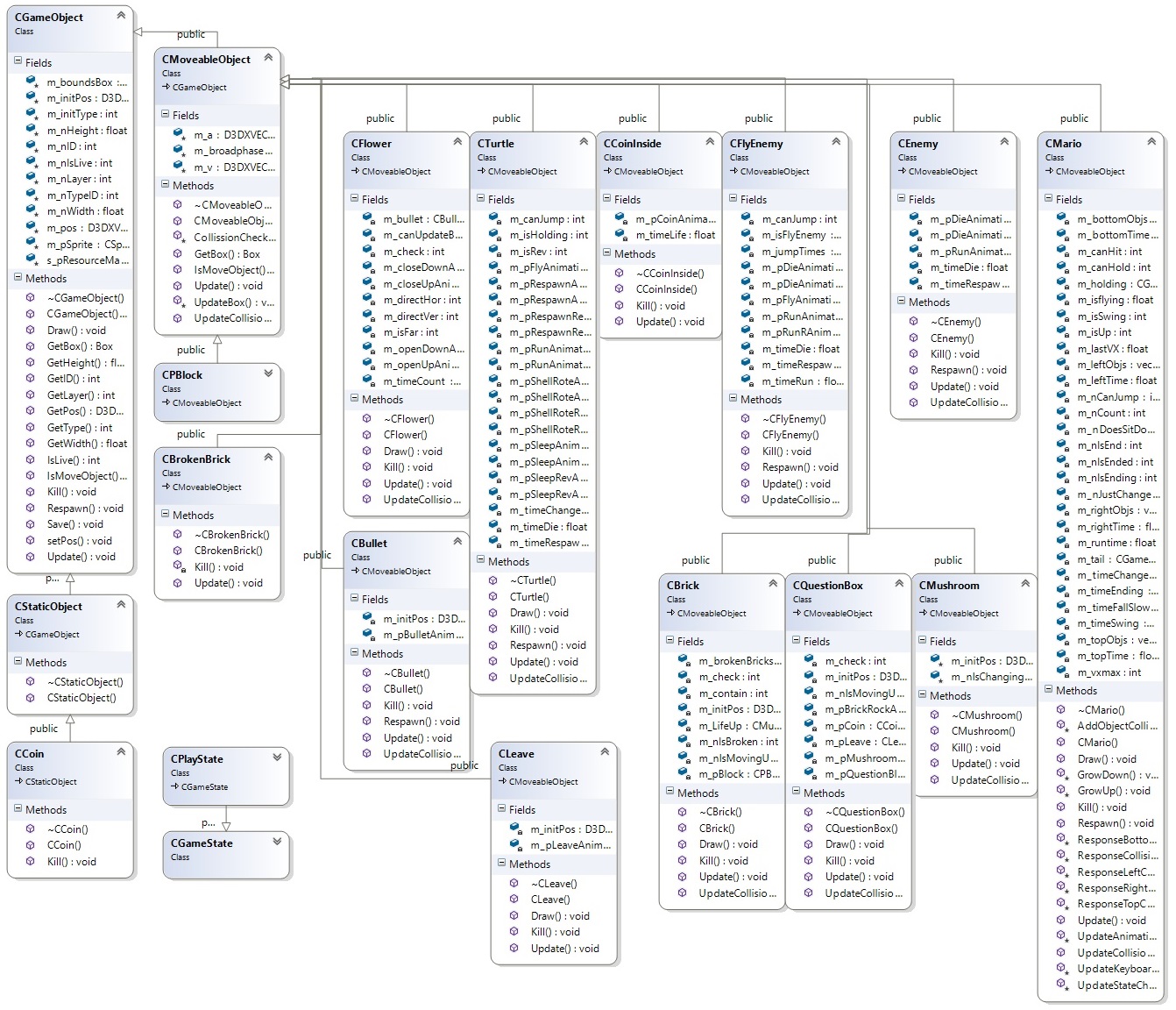
# Phân tích và thiết kế

## Phân tích

### Danh sách các class sử dụng trong game

|  |  |  |
| --- | --- | --- |
| **STT** | **Tên Class** | **Chức Năng** |
| 1 | CGameObject | Đây là lớp cha, chứa các hàm ảo và các thuộc tính cần thiết cho việc lưu trữ, tất cả các đối tượng trong game đều kế thừa từ lớp này. |
| 2 | CMoveableObject | Đây là lớp cha của tất cả các đối tượng có thể di chuyển, kế thừa từ lớp CGameObject |
| 3 | CStaticObject | Đây là lớp cha của tất cả các đối tượng tĩnh, kế thừa từ lớp CGameObject |
| 4 | CMario | Chứa các hàm xử lý và hiển thị mario |
| 5 | CEnemy | Lớp thể hiện con quái hình nấm |
| 6 | CFlyEnemy | Lớp thể hiện con quái hình nấm có cánh |
| 7 | CTurtle | Lớp thể hiện con rùa và con rùa có cánh. |
| 8 | CFlower | Lớp thể hiện bông hoa bắn đạn và bông hoa ăn thịt. |
| 9 | CBullet | Lớp thể hiện đạn của bông hoa |
| 10 | CBrick | Lớp thể hiện viên gạch có thể đập vỡ được |
| 11 | CBrokenBrick | Lớp thể hiện các mảnh vỡ của viên gạch |
| 12 | CQuestionBox | Lớp thể hiện khối dấu hỏi |
| 13 | CMushroom | Lớp thể hiện nấm đỏ (nấm tăng kích thước) và nấm xanh (nấm mạng). |
| 14 | CLeave | Lớp thể hiện chiếc lá |
| 15 | CCoin | Lớp thể hiện đồng tiền |
| 16 | CCoinInside | Lớp thể hiện đồng tiền bên trong khối dấu hỏi |
| 17 | CPBlock | Lớp thể hiện khối chữ P |
| 18 | CGameApp | Lớp điều khiển mọi hoạt động của game thông qua việc chuyển đổi qua lại các state |
| 19 | CGameState | Lớp nền cho các state trong game như menu state, play state…. |
| 20 | CPlayState | Lớp chạy chính của game khi chơi. |
| 21 | CGraphic | Lớp hỗ trợ cho việc dựng hình trong game sử dụng thư viện đồ họa Direct3D của DirectX |
| 22 | CInput | Lớp quản lý thiết bị nhập trong game |
| 23 | CQNode | Lớp node trong cấu trúc cây tứ phân |
| 24 | CTreeObject | Lớp các đối tượng bên trong node của cây |
| 25 | CSprite | Lớp xử lý cho các sprite trong game |
| 26 | CTexture | Lớp xử lý cho các texture (hình ảnh tạo nên các sprite) |
| 27 | CViewport | Lớp quản lý viewport, giúp xác định khung cảnh nào của thế giới được vẽ lên màn hình. |
| 28 | CSoundManager | Lớp xử lý âm thanh trong game |
| 29 | CPlayResourceManager | Lớp quản lý tài nguyên trong game bao gồm texture, sprite, âm thanh. |
| 30 | CWindow | Lớp hỗ trợ việc khởi tạo cửa sổ |

### Sơ đồ các class kế thừa trong game



Hình 1. Sơ đồ các class kế thừa

## Thiết kế:

### Thiết kế chi tiết các class quản lý đối tượng trong game

#### Class CGameObject (lớp cha, chủ yếu chứa các biến dùng chung cho các class dẫn xuất và các hàm ảo)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên Hàm/ Tên Biến** | **Kiểu Dữ Liệu / Kiểu Trả Về** | **Chức Năng** |
| 1 | m\_nID | int | ID của đối tượng |
| 2 | m\_nboundBox | Box | Khung va chạm |
| 3 | m\_pos | D3DXVECTOR2 | Vị trí |
| 4 | m\_pSprite | CSprite\* | Sprite của đối tượng |
| 5 | m\_nHeight | float | Chiều cao |
| 6 | m\_nWidth | float | Chiều rộng |
| 7 | m\_nTypeID | int | Kiểu đối tượng |
| 8 | m\_nLayer | int | Layer của đối tượng |
| 9 | m\_nIsLive | int | Còn sống hay không |
| 10 | s\_pResourceManager | CPlayResourceManager\* | Đối tượng quản lý resource |
| 11 | CGameObject(D3DXVECTOR2 pos, int id, int typeID, int layer) |  | Hàm khởi tạo |
| 12 | Update(float timeFrame) | Void | Hàm ảo |
| 13 | Draw(CViewport \*pViewport) | Void | Hàm ảo |
| 14 | setPos(float x, float y) | Void | Set vị trí cho đối tượng |
| 15 | GetLayer() | int | Lấy layer |
| 16 | GetType() | int | Lấy loại đối tượng |
| 17 | GetID() | int | Lấy ID |
| 18 | GetHeight() | float | Lấy chiều cao |
| 19 | GetWidth() | float | Lấy chiều rộng |
| 20 | GetPos() | D3DXVECTOR2 | Lấy vị trí |
| 21 | GetBox() | Box | Lấy khung va chạm |
| 22 | IsMoveObject() | int | Kiểm tra phải đối tượng di chuyển được hay không |
| 23 | IsLive() | int | Kiểm tra còn sống hay không |
| 24 | ~CGameObject(void) |  | Hàm ảo – hủy đối tượng |

#### Class CMoveableObject (Lớp cha của các đối tượng động)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên Hàm/ Tên Biến** | **Kiểu Dữ Liệu / Kiểu Trả Về** | **Chức Năng** |
| 1 | m\_a | D3DXVECTOR2 | Gia tốc |
| 2 | m\_v | D3DXVECTOR2 | Vận tốc |
| 3 | m\_broadphaseBox | Box | Khung broadphase dùng trong xét va chạm |
| 4 | CMoveableObject(D3DXVECTOR2 pos, int id, int typeID, int layer); |  | Hàm khởi tạo đối tượng động |
| 5 | CollissionCheck(CMoveableObject\* moveObject, float &imefr, float &normaly, float imeframe) | float | Xét va chạm với đối tượng tĩnh |
| 6 | CollissionCheck(CStaticObject\* staticObject, float &imefr, float &normaly, float imeframe) | float | Xét va chạm với đối tượng tĩnh |
| 7 | UpdateBox(float imeframe) | void | Cập nhật khung va chạm |
| 8 | UpdateCollision(play::CQNode\* quadTree, float imeframe) | void | Hàm ảo – xét va chạm và update vị trí đối tượng dựa trên va chạm |
| 9 | Update(float timeframe) | void | Hàm ảo – update các thuộc tính của đối tượng như vận tốc, gia tốc, sprite.. |
| 10 | GetBox() | Box | Lấy khung va chạm |
| 11 | IsMoveObject() | int | Hàm ảo – kiểm tra có phải đối tượng di chuyển được hay không |
| 12 | ~CStaticObject(void) |  | Hàm ảo – hủy đối tượng |

#### Class CStaticObject (Lớp các đối tượng tĩnh)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên Hàm/ Tên Biến** | **Kiểu Dữ Liệu / Kiểu Trả Về** | **Chức Năng** |
| 1 | CStaticObject(D3DXVECTOR2 pos, int id, int typeID, int layer) |  | Hàm khởi tạo đối tượng tĩnh |
| 2 | ~CStaticObject(void) |  | Hàm hủy đối tượng tĩnh |

#### Class CMario (Quản lý Mario)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên Hàm/ Tên Biến** | **Kiểu Dữ Liệu / Kiểu Trả Về** | **Chức Năng** |
| 1 | m\_lastVX | float | Vận tốc mario trước khi dừng |
| 2 | m\_timeChange | float | Dùng khi mario thay đổi trạng thái từ lớn thành nhỏ |
| 3 | m\_timeEnding | float | Thời gian tự cập nhật lúc kết thúc màn |
| 4 | m\_timeFallSlow | float | Thời gian rơi chậm của mario |
| 5 | m\_timeSwing | float | Thời gian quất đuôi của mario |
| 6 | m\_ncanJump | int | Xác định mario có nhảy được hay không |
| 7 | m\_nJustChange | int | Xác định có phải mario mới chuyển trạng thái hay không |
| 8 | m\_nDoesSitDown | int | Mario có đang ngồi hay không |
| 9 | m\_isSwing | int | Mario có đang quất đuôi hay không |
| 10 | m\_isUp | int | Mario có đang nhảy hay không |
| 11 | m\_canHold | int | Mario có thể ôm rùa hay không |
| 12 | m\_canHit | int | Mario có thể quất đuôi làm chết quái hay không |
| 13 | m\_runtime | float | Thời gian chạy trước khi chuyển sang chạy nhanh |
| 14 | m\_isflying | int | Mario có đang bay hay không |
| 15 | m\_vxmax | int | Vận tốc cực đại theo phương x |
| 16 | m\_nIsEnded | int | Kết thúc màn hay chưa |
| 17 | m\_nIsEnding | int | Có đang kết thúc màn hay không |
| 18 | m\_nCount | int | Dùng để update sprite lúc mario chuyển trạng thái |
| 19 | GrowUp() | void | Mario lớn lên |
| 20 | GrowDown() | void | Mario nhỏ xuống |
| 21 | UpdateStateChanging(float timeFrame) | void | Cập nhật trạng thái mario lúc mới chuyển trạng thái |
| 22 | UpdateKeyboard(CInput\* pInput, float timeFrame) | void | Cập nhật bàn phím |
| 23 | UpdateAnimation(float timeFame) | void | Cập nhật hiệu ứng sprite cho mario |
| 24 | ResponseCollision(float timeFrame) | void | Phản ứng lại các va chạm |
| 25 | ResponseLeftCollision(float timeFrame) | int | Phản ứng lại các va chạm bên trái |
| 26 | ResponseRightCollision(float timeFrame) | int | Phản ứng lại các va chạm bên phải |
| 27 | ResponseTopCollision(float timeFrame) | int | Phản ứng lại các va chạm ở trên |
| 28 | ResponseBottomCollision(float timeFrame) | int | Phản ứng lại các va chạm ở dưới |
| 29 | AddObjectCollision(float timeFrame, float collisionTime, float normalx, float normaly, CGameObject\* pObject) | void | Thêm các đối tượng va chạm vào list tương ứng. |
| 30 | CMario(D3DXVECTOR2 posint,int id, int typeID, int layer) |  | Phương thức khởi tạo maro |
| 31 | ~CMario(); |  | Phương thức hủy |
| 32 | Draw(CViewport\* pViewport) | void | Phương thức vẽ |
| 33 | UpdateCollision(play::CQNode\* quadTree, float timeFrame) | void | Phương thức cập nhật vận tốc, gia tốc |
| 34 | Kill(int type = 0) | void | Giết chết mario |

#### Class CBrick (Viên gạch có thể đập vỡ)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_initPos | D3DXVECTOR2 | Vị trí ban đầu của gạch |
| 2 | m\_brokenBrick | CBrokenBrick\* | Mảng lưu trữ 4 mảnh gạch vỡ |
| 3 | m\_nIsMovingUp | int | Xác định gạch có đang nảy lên |
| 4 | m\_nIsBroken | int | Xác định gạch vỡ hay chưa |
| 5 | m\_pBlock | CPBlock\* | Con trỏ đến khối chữ P |
| 6 | m\_contain | int | Xác định khối gạch chứa chữ P hay nấm mạng |
| 7 | m\_check | int | Biến kiểm tra |
| 8 | m\_LifeUp | CMushroom\* | Con trỏ đến nấm mạng |
| 9 | CBrick(D3DXVECTOR2 pos, int id, int typeID, int layer) |  | Hàm khởi tạo |
| 10 | ~CBrick() |  | Hàm hủy |
| 11 | Kill(int type = 0) | void | Thay đổi trạng thái viên gạch |
| 12 | Update(float timeFrame) | void | Cập nhật vận tốc, gia tốc, sprite |
| 13 | UpdateCollision(play::CQNode\* quadTree, float timeFrame) | void | Cập nhật va chạm, vị trí |
| 14 | Draw(CViewport\* pViewport) | void | Phương thức vẽ |

#### Class CEnemy (con quái hình nấm)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_pDieAnimation1 | CSprite\* | Sprite hiệu ứng chết 1 |
| 2 | m\_pDieAnimation2 | CSprite\* | Sprite hiệu ứng chết 2 |
| 3 | m\_pRunAnimation | CSprite\* | Sprite hiệu ứng chạy |
| 4 | m\_timeDie | float | Thời gian chết của enemy |
| 5 | CEnemy(D3DXVECTOR2 pos, int id, int typeID, int layer) |  | Hàm khởi tạo |
| 6 | ~CEnemy() |  | Hàm hủy |
| 7 | UpdateCollision(play::CQNode\* quadTree, float timeFrame) | void | Phương thức cập nhật va chạm, vị trí |
| 8 | Update(float timeFrame) | void | Phương thức cập nhật vận tốc, gia tốc, sprite |
| 9 | Kill(int type = 0) | void | Phương thức giết chết enemy |

#### Class CFlyEnemy (con quái hình nấm có cánh)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_pDieAnimation1 | CSprite\* | Sprite hiệu ứng chết 1 |
| 2 | m\_pDieAnimation2 | CSprite\* | Sprite hiệu ứng chết 2 |
| 3 | m\_pRunAnimation | CSprite\* | Sprite hiệu ứng chạy1 |
| 4 | m\_pRunRAnimation | CSprite\* | Sprite hiệu ứng chạy 2 |
| 5 | m\_pFlyAnimation | CSprite\* | Sprite hiệu ứng bay |
| 6 | m\_timeDie | float | Thời gian chết của enemy |
| 7 | CEnemy(D3DXVECTOR2 pos, int id, int typeID, int layer) |  | Hàm khởi tạo |
| 8 | ~CEnemy() |  | Hàm hủy |
| 9 | UpdateCollision(play::CQNode\* quadTree, float timeFrame) | void | Phương thức cập nhật va chạm, vị trí |
| 10 | Update(float timeFrame) | void | Phương thức cập nhật vận tốc, gia tốc, sprite |
| 11 | Kill(int type = 0) | void | Phương thức giết chết enemy |

#### Class CTurtle (con rùa - rùa xanh, rùa đỏ và rùa bay)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_pRunAnimationR | CSprite\* | Sprite hiệu ứng chạy của rùa đỏ |
| 2 | m\_pSleepAnimationR | CSprite\* | Sprite hiệu ứng ngủ của rùa đỏ |
| 3 | m\_pSleepRevAnimationR | CSprite\* | Sprite hiệu ứng ngủ ngược của rùa đỏ |
| 4 | m\_pShellRoteAnimationR | CSprite\* | Sprite hiệu ứng xoay mai rùa đỏ |
| 5 | m\_pShellRoteRevAnimationR | CSprite\* | Sprite hiệu ứng xoay mai rùa đỏ bị ngược |
| 6 | m\_pRespawnAnimationR | CSprite\* | Sprite hiệu ứng hồi sinh của rùa đỏ |
| 7 | m\_pRespawnRevAnimationR | CSprite\* | Sprite hiệu ứng hồi sinh của rùa đỏ bị ngược |
| 8 | m\_pRunAnimationG | CSprite\* | Sprite hiệu ứng chạy của rùa xanh |
| 9 | m\_pSleepAnimationG | CSprite\* | Sprite hiệu ứng ngủ của rùa xanh |
| 10 | m\_pSleepRevAnimationG | CSprite\* | Sprite hiệu ứng ngủ ngược của rùa xanh |
| 11 | m\_pShellRoteAnimationG | CSprite\* | Sprite hiệu ứng xoay mai rùa xanh |
| 12 | m\_pShellRoteRevAnimationG | CSprite\* | Sprite hiệu ứng xoay mai rùa xanh bị ngược |
| 13 | m\_pRespawnAnimationG | CSprite\* | Sprite hiệu ứng hồi sinh của rùa xanh |
| 14 | m\_pRespawnRevAnimationG | CSprite\* | Sprite hiệu ứng hồi sinh của rùa xanh bị ngược |
| 15 | m\_pFlyAnimation | CSprite\* | Sprite hiệu ứng rùa xanh bay |
| 16 | m\_isRev | int | Xác định rùa có bị ngược hay không |
| 17 | m\_isHolding | int | Xác định rùa có bị mario ôm hay không |
| 18 | m\_canJump | int | Có thể nhảy hay không |
| 19 | m\_timeDie | float | Thời gian chết |
| 20 | m\_timeRespawn | float | Thời gian để hồi sinh |
| 21 | m\_initPos | D3DXVECTOR2 | Vị trí ban đầu |
| 22 | m\_initType | int | Loại đối tượng ban đầu |
| 23 | m\_timeChange | float | Thời gian chuyển đổi trạng thái |
| 24 | Respawn() | void | Phương thức hồi sinh rùa |
| 25 | CTurtle(D3DXVECTOR2 pos, int id, int typeID, int layer) |  | Hàm khởi tạo |
| 26 | ~CTurtle(void) |  | Hàm hủy |
| 27 | UpdateCollision(play::CQNode\* quadTree, float timeFrame) | void | Phương thức cập nhật va chạm, vị trí |
| 28 | Update(float timeFrame) | void | Phương thức cập nhật vận tốc, gia tốc, sprite |
| 29 | Kill(int type = 0) | void | Phương thức giết chết rùa |
| 30 | Draw(CViewport\* pViewport) | void | Phương thức vẽ |

#### Class CFlower (bông hoa - hoa bắn đạn và hoa ăn thịt)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_closeDownAnimation | CSprite\* | Sprite hiệu ứng đóng mở bông hoa hướng xuống |
| 2 | m\_openDownAnimation | CSprite\* | Sprite hiệu ứng mở bông hoa hướng xuống |
| 3 | m\_closeUpAnimation | CSprite\* | Sprite hiệu ứng đóng mở bông hoa hướng lên |
| 4 | m\_openUpAnimation | CSprite\* | Sprite hiệu ứng mở bông hoa hướng lên |
| 5 | m\_initPos | D3DXVECTOR2 | Vị trí ban đầu |
| 6 | m\_timeCount | int | Thời gian dừng lại của bông hoa |
| 7 | m\_directHor | int | Hướng bắn đạn theo chiều ngang |
| 8 | m\_directVer | float | Hướng bắn đạn theo chiều dọc |
| 9 | m\_isFar | float | Xác định bắn đạn xa hay gần |
| 10 | m\_canUpdateBullet | D3DXVECTOR2 | Có thể update hướng đạn hay không |
| 11 | m\_check | int | Biến kiểm tra |
| 12 | m\_bullet | CBullet\* | Con trỏ đến viên đạn |
| 13 | CFlower(D3DXVECTOR2 pos, int id, int typeID, int layer) |  | Hàm khởi tạo |
| 14 | ~CFlower() |  | Hàm hủy |
| 15 | UpdateCollision(play::CQNode\* quadTree, float timeFrame) | void | Phương thức cập nhật va chạm, vị trí |
| 16 | Update(float timeFrame) | void | Phương thức cập nhật vận tốc, gia tốc, sprite |
| 17 | Kill(int type = 0) | void | Phương thức giết chết bông hoa |
| 18 | Draw(CViewport\* pViewport) | void | Phương thức vẽ |

#### Class CBullet (Đạn của bông hoa)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_pBulletAnimation | CSprite\* | Sprite hiệu ứng viên đạn |
| 5 | m\_initPos | D3DXVECTOR2 | Vị trí ban đầu |
| 13 | CBullet(int direct, D3DXVECTOR2 pos, int isTail) |  | Hàm khởi tạo |
| 14 | ~CBullet() |  | Hàm hủy |
| 15 | UpdateCollision(play::CQNode\* quadTree, float timeFrame) | void | Phương thức cập nhật va chạm, vị trí |
| 16 | Update(float timeFrame) | void | Phương thức cập nhật vận tốc, gia tốc, sprite |
| 17 | Kill(int type = 0) | void | Phương thức giết chết bông hoa |
| 18 | Respawn(int direct) | void | Phương thức hồi sinh đạn |

#### Class CBrokenBrick (mảnh vỡ gạch)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 13 | CBrokenBrick(D3DXVECTOR2 pos, int type) |  | Hàm khởi tạo |
| 14 | ~CBrokenBrick() |  | Hàm hủy |
| 16 | Update(float timeFrame) | void | Phương thức cập nhật vận tốc, gia tốc |
| 17 | Kill(int type = 0) | void | Phương thức làm mất mảnh vỡ |

#### Class CQuestionBox (khối dấu hỏi)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_initPos | D3DXVECTOR2 | Vị trí ban đầu của gạch |
| 2 | m\_nIsMovingUp | int | Xác định khối có đang nảy lên |
| 3 | m\_check | int | Biến kiểm tra |
| 4 | m\_pBrickRockAnimation | CSprite\* | Sprite biến thành đá |
| 5 | m\_pQuestionBlockAnimation | CSprite\* | Sprite dấu hỏi |
| 6 | m\_pCoin | CCoinInside\* | Con trỏ đến đồng tiền bên trong |
| 7 | m\_pMushroom | CMushroom\* | Con trỏ đến nấm tăng kích thước |
| 8 | m\_pLeave | CLeave\* | Con trỏ đến chiếc lá |
| 9 | CQuestionBox(D3DXVECTOR2 pos, int id, int typeID, int layer) |  | Hàm khởi tạo |
| 10 | ~CQuestionBox() |  | Hàm hủy |
| 11 | Update(float timeFrame) | void | Cập nhật vận tốc, gia tốc, sprite |
| 12 | UpdateCollision(play::CQNode\* quadTree, float timeFrame) | void | Cập nhật va chạm, vị trí |
| 13 | Draw(CViewport\* pViewport) | void | Phương thức vẽ |

#### Class CMushroom (nấm mạng và nấm tăng kích thước)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_nIsChangingState | int | Nấm có đang đi lên hay không |
| 2 | m\_initPos | D3DXVECTOR2 | Vị trí ban đầu |
| 3 | CMushroom(D3DXVECTOR2 pos, int typeID) |  | Hàm khởi tạo |
| 4 | ~CMushroom() |  | Hàm hủy |
| 5 | UpdateCollision(play::CQNode\* quadTree, float timeFrame) | void | Phương thức cập nhật va chạm, vị trí |
| 6 | Update(float timeFrame) | void | Phương thức cập nhật vận tốc, gia tốc, sprite |
| 7 | Kill(int type = 0) | void | Phương thức làm mất nấm |

#### Class CCoin (Đồng xu)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 3 | CCoin(D3DXVECTOR2 pos, int id, int typeID, int layer) |  | Hàm khởi tạo |
| 4 | ~CCoin(void) |  | Hàm hủy |
| 7 | Kill(int type = 0) | void | Phương thức làm mất đồng tiền |

#### Class CLeave (Chiếc lá)

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_pLeaveAnimation | CSprite\* | Sprite chiếc lá |
| 2 | m\_initPos | D3DXVECTOR2 | Vị trí ban đầu |
| 3 | CLeave(D3DXVECTOR2 pos) |  | Hàm khởi tạo |
| 4 | ~CLeave(); |  | Hàm hủy |
| 5 | Update(float timeFrame) | void | Phương thức cập nhật vận tốc, gia tốc, sprite |
| 6 | Draw(CViewport\* pViewport) | void | Phương thức vẽ lá |

#### Class CCoinInside (đồng xu bên trong khối dấu hỏi):

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_pCoinAnimation | CSprite\* | Sprite đồng tiền |
| 2 | m\_timeLife | float | Thời gian sống của đồng tiền |
| 3 | CCoinInside(D3DXVECTOR2 pos) |  | Hàm khởi tạo |
| 4 | ~CCoinInside() |  | Hàm hủy |
| 5 | Update(float timeFrame) | void | Phương thức cập nhật vận tốc, gia tốc, sprite |
| 6 | Draw(CViewport\* pViewport) | void | Phương thức vẽ |
| 7 | Kill() | void | Phương thức làm mất đồng tiền |

#### Class CPBlock : khối chữ P

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_initPos | D3DXVECTOR2 | Vị trí ban đầu của gạch |
| 2 | m\_nIsMovingUp | int | Xác định khối có đang đi lên |
| 3 | m\_isPressed | int | Biến kiểm tra chữ P nhấn xuống chưa |
| 9 | CPBlock(D3DXVECTOR2 pos, int id, int typeID, int layer) |  | Hàm khởi tạo |
| 10 | ~CPBlock() |  | Hàm hủy |
| 11 | Update(float timeFrame) | void | Cập nhật vận tốc, gia tốc, sprite |
| 12 | UpdateCollision(play::CQNode\* quadTree, float timeFrame) | void | Cập nhật va chạm, vị trí |
| 13 | Kill(int type = 0) | void | Phương thức nhấn chữ P xuống |

#### Class CGameApp (Lớp ứng dụng game):

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_szMapFileName | wchar\_t\* | Đường dẫn của file map sẽ được load |
| 2 | m\_nCurGameStateID | int | ID trạng thái hiện tại của ứng dụng game |
| 3 | m\_pCurGameState | CGameState\* | Con trỏ đến thể hiện của state hiện tại |
| 4 | CGameApp(HINSTANCE hInstance) |  | Hàm khởi tạo |
| 5 | ~CGameApp() |  | Hàm hủy |
| 6 | Run() | void | Phương thức chạy chính điều khiển toàn bộ chương trình |
| 7 | ChangeToStateHandle(int nGameStateID) | void | Phương thức chuyển state |

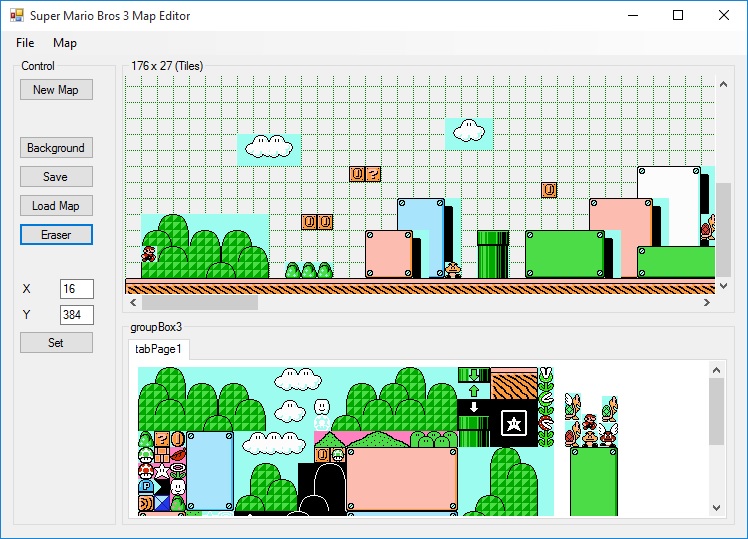
#### Class CGameState : lớp nền cho các state trong game

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **Tên hàm/ tên biến** | **Kiểu dữ liệu/Kiểu trả về** | **Chức năng** |
| 1 | m\_pGameApp | CGameApp\* | Con trỏ đến đối tượng GameApp |
| 2 | m\_pResourceManager | CPlayResourceManager\* | Con trỏ đến đối tượng quản lý resource |
| 3 | m\_nNextState | int | State tiếp theo |
| 4 | m\_nIsDone | int | State đã chạy xong chưa |
| 5 | CGameState(CGameApp\* pGameApp) |  | Hàm khởi tạo |
| 6 | ~CGameState() |  | Hàm hủy |
| 7 | Run() = 0 | void | Hàm thuần ảo – Phương thức chạy của state |
| 8 | Draw() = 0 | void | Hàm thuần ảo – Phương thức vẽ |
| 9 | GetResourceManager() | CPlayResourceManager\* | Phương thức trả về resource manager của game hiện tại |

### Thiết kế map

#### Cách thức thực hiện

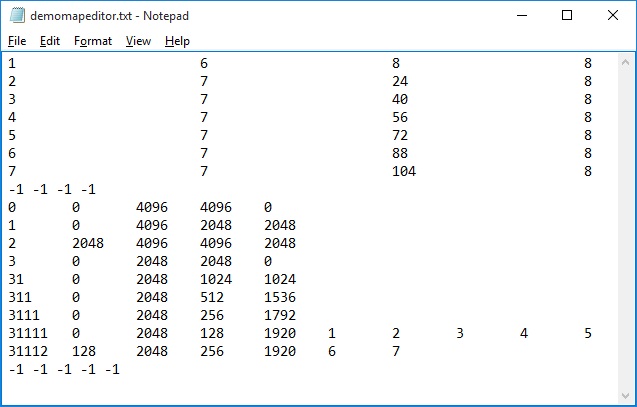
Sử dụng công cụ hỗ trợ làm map viết bằng ngôn ngữ C#, với chức năng chính là hỗ trợ đưa các đối tượng vào map, canh chỉnh vị trí cho phù hợp và sau đó lưu file map dưới định dạng text. Công cụ đã được cài đặt sẵn tính năng buid wadtree trước khi lưu. Về phần các đối tượng, xem mỗi đối tượng vẽ là một game object, có ID, TypeID (loại đối tượng), X, Y, Width, Height để thuận tiện cho việc load map trong game.



Hình 2. Công cụ map editor do nhóm thiết kế

#### Kết quả sau khi lưu map

Kết quả sau khi lưu ta sẽ có một file map dạng text, với định dạng như sau:



Hình 3. Demo cấu trúc file map sau khi thiết kế

Cấu trúc file lưu được chia thành 2 phần, ngăn cách nhau bởi dãy số -1, phần trên lưu danh sách tất cả các đối tượng có trong map, nội dung lưu theo thứ tự: ID, TypeID (loại đối tượng), X, Y. Phần bên dưới là phần có được sau khi build wadtree, nội dung lưu theo thứ tự: NodeID, X1, Y1, X2, Y2 và sau đó là danh sách các đối tượng của node, nếu không có gì tức là node đó rỗng không có đối tượng.

# Các kiến thức áp dụng

## Phương pháp lập trình lập hướng đối tượng

### Khái niệm

Lập trình hướng đối tượng (gọi tắt là OOP, từ chữ Anh ngữ object-oriented programming), hay còn gọi là lập trình định hướng đối tượng, là kĩ thuật lập trình hỗ trợ công nghệ đối tượng. OOP được xem là giúp tăng năng suất, đơn giản hóa độ phức tạp khi bảo trì cũng như mở rộng phần mềm bằng cách cho phép lập trình viên tập trung vào các đối tượng phần mềm ở bậc cao hơn. Ngoài ra, nhiều người còn cho rằng OOP dễ tiếp thu hơn cho những người mới học về lập trình hơn là các phương pháp trước đó.

### Đặc điểm của lập trình hướng đối tượng

Tập trung vào dữ liệu thay cho các hàm. Chương trình được chia thành các đối tượng độc lập. Cấu trúc dữ liệu được thiết kế sao cho đặc tả được các đối tượng. Dữ liệu được che giấu, bao bọc. Các đối tượng trao đổi với nhau thông qua các hàm. Chương trình được thiết kế theo hướng tiếp cận từ dưới lên.

### Trừu tượng hóa dữ liệu

Khi bắt đầu phát triển phần mềm, lập trình viên sẽ không đi vào viết lệnh ngay lập tức. Công việc đầu tiên cần nghiên cứu dữ án, phân tích các đặc tả phần mềm, các thành phần cần có. Theo đó, lập trình viên sẽ phải xác định những thông tin, dữ liệu cần thiết cho từng thành phần. Như vậy, trừu tượng hóa dữ liệu là quá trình xác định, tập hợp những thuộc tính, hành xử cho một thực thể cụ thể liên quan trong ứng dụng. Trong quá trình đó, có thể có những chi tiết sẽ được cài đặt nhưng ngược lại sẽ có những chi tiết cần loại bỏ vì chúng không nằm trong yêu cầu.

Những ưu điểm của việc trừu tượng hóa dữ liệu bao gồm:

* Tập trung vào vấn đề.
* Xác định rõ những thành phần và chức năng cần thiết.
* Loại trừ những chi tiết không cần thiết.

### Tính đa hình

Đa hình là khả năng cho phép một hoạt động có những hành xử khác nhau trong cùng một đối tượng. khái niệm này ám chỉ việc đối tượng có nhiều phương thức cùng tền nhưng lại thực hiện những nhiệm vụ khác nhau. Trình biên dịch và hệ thống thực thi sẽ so khớp mỗi lời gọi phương thức, đối tượng. Khả năng tìm kiếm đúng để sử dụng trong những tình huống phức tạp là bản chất của tính đa hình. Đa hình là một trong những đặc trưng quan trọng của hệ thông hướng đối tượng.

### Tính kế thừa

Kế thừa là khả năng cho phép một lớp chia sẻ thuộc tính, hành xử được cài đặt trong nó. Nguyên lý kế thừa hỗ trợ việc tạo ra cấu trúc phần cấp các lớp. Nó được hiểu như một cở chế sao chép ảo không đơn điệu và có chắt lọc. Một lớp có thể thừa kế từ một hay nhiều lớp khác. Theo đó, lớp được kế thừa là lớp cha và lớp kế thừa là lớp con.

### Tính đóng gói dữ liệu

Đóng gói dữ liệu là quá trình che dấu những cài đặt cụ thể của một đối tượng với một đối tượng sử dụng khác. Dữ liệu được tổ chức có thể phơi bày với thế giới bên ngoài hoặc che dấu vào bên trong đối tượng. Những thuộc tính, hành xử được phởi bày để được truy cập, triệu gọi từ một đối tượng khác. Ngược lại, những thuộc tính, hành xử được che giấu chỉ đảm nhận hành xử hay lưu trữ dữ liệu cho chính đối tượng đó, chúng không có khả năng truy cập từ ngoài vào.

### Những ưu điểm của lập trình hướng đối tượng.

Loại bỏ được những dư thừa, trùng lặp trong việc xây dựng ứng dụng. Cài đặt đối tượng giúp xức tiến việc sử dụng lại, trao đổi giữa các đối tượng với nhau do đó sẽ giảm kích thước, thời gian xử lý,… thời gian phát triển hệ thống, tăng năng xuất lao động. Dễ bảo trì, nâng cấp, giảm lỗi.

### Các thành phần trong lập trình hướng đối tượng

- class: các chức năng, thành phần của một thực thể được định nghĩa một cách chung nhất trong một mẫu cụ thể được gọi là class.

- Object: là một đối tượng cụ thể của class

- Attribute: là những thành phần được định nghĩa trong một class.

- Operation: giống như một dịch vụ được triệu gọi bởi một đối tượng

- Method: còn gọi là phương thức định nghĩa cụ thể một operation

-Message: là những đòi hỏi, những thông tin cần có cho một operation.

- Event: là những kích hoạt để thực thi một hay nhiều operation

- Construction: là triệu gọi đầu tiên khi một object được tạo. Constructor là một method đặc biệt được gọi đầu tiên trước khi một method hay attribute có thể được sử dụng trong một object của một class. Constructor có thể khỏi gán nhữn thuộc tính, cấp phát bộ nhớ nếu cần thiết.

- Destruction: là phương thức đặc biệt được gọi để hủy một object. Destruction sẽ xóa object, giải phóng bộ nhớ mà constructor cấp phát. Theo đó, mọi khả năng sử dụng truy cập tới object này sẽ không còn.

## Hướng đối tượng trong game super mario bros 3

Nguyên tắc kế thừa cho phép các lớp dẫn xuất sử dụng các phương thức từ lớp cơ sở như những phương thức của chính nó, mà không cần thiết phải định nghĩa lại.

Để thuận tiện cho việc tái sử dụng lại các phương thức hay dùng trong các đối tượng khác nhau, việc tự động so khớp và thực thi các phương thức và các đối tượng. Trong game, chúng em sử dụng 1 class CGameObject là class cha và các class dẫn xuất.

Việc sử dụng chung con trỏ của lớp cha và tự động trỏ đến đúng đối tượng dẫn xuất giúp cho việc xử lý trở nên dễ dàng hơn.

Mỗi vật trong game được đưa vào một class quản lý. Việc chia nhỏ này giúp cho việc quản lý cũng như xử lý trong game dễ dàng hơn.

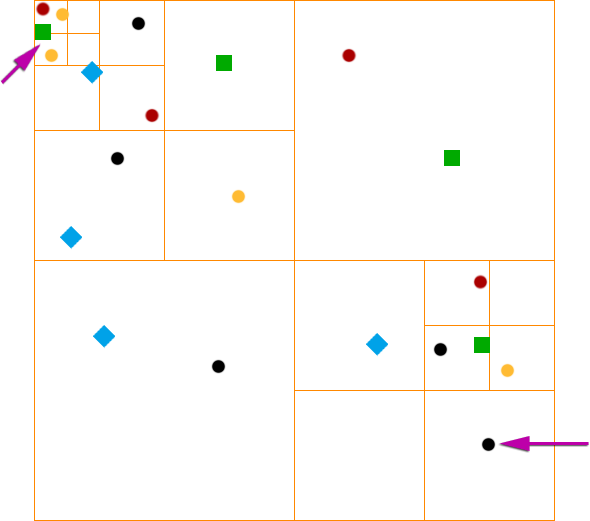
Danh sách các class sử dụng kế thừa trong game

|  |  |  |
| --- | --- | --- |
| **STT** | **Tên Class** | **Chức Năng** |
| 1 | CGameObject | Đây là lớp cha, chứa các hàm ảo và các thuộc tính cần thiết cho việc lưu trữ, tất cả các đối tượng trong game đều kế thừa từ lớp này. |
| 2 | CMoveableObject | Đây là lớp cha của tất cả các đối tượng có thể di chuyển, kế thừa từ lớp CGameObject |
| 3 | CStaticObject | Đây là lớp cha của tất cả các đối tượng tĩnh, kế thừa từ lớp CGameObject |
| 4 | CGameState | Lớp nền cho các state trong game như menu state, play state…. |

## Kỹ thuật phân chia không gian bằng QuadTree

### Giới thiệu về cây tứ phân (QuadTree)

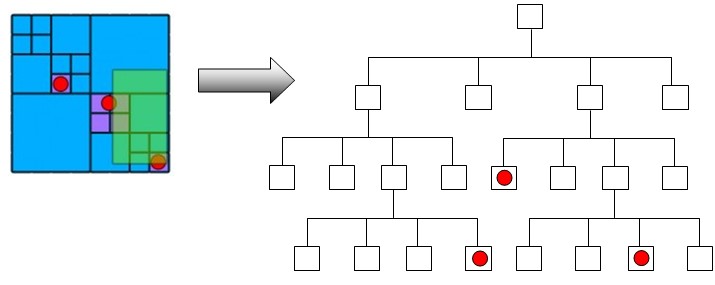
* Quadtree là một cấu trúc dữ liệu phân nhánh dạng cây, được sử dụng để phân hoạch vùng không gian hai chiều hiện tại thành các vùng nhỏ hơn và dễ quản lý hơn. Khác với cây nhị phân, mỗi lần phân hoạch, quadtree sẽ chia vùng hiện tại thành 4 vùng. Do đó mỗi node trong quadtree có thể có 4 node dẫn xuất hoặc không có node dẫn xuất nào, không có ngoại lệ.
* Việc sử dụng Quadtree giúp chúng ta quản lý các đối tượng trong game một cách hiệu quả hơn, bằng cách phân chia vùng cần xử lý các đối tượng thành những vùng con, các đối tượng sẽ được đưa vào các vùng tương ứng và được quản lý riêng biệt. Quá trình phân chia cứ thế tiếp diễn cho đến khi mỗi vùng chỉ chứa một số lượng đối tượng nhất định hoặc mức độ phân chia đạt mức “chấp nhận được”.



*Hình 1.1.1 - Sử dụng cây tứ phân trong phân hoạch không gian*

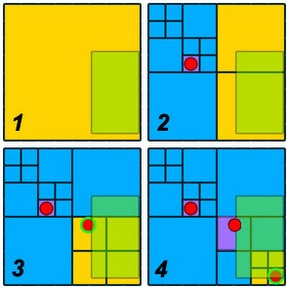
### Sử dụng Quadtree để phân hoạch không gian trong game

* Một **vấn đề** lớn khi làm game mario cũng như một số game 2d khác đó là số lượng các đối tượng trong một màn chơi là khá lớn (khoảng 500). Nếu khi cần xét va chạm, mà ta phải duyệt qua tất cả các đối tượng thì sẽ rất lãng phí tài nguyên.Trong khi đó, hầu như ta chỉ cần xét va chạm với các đối tượng trong màn hình, như vậy việc xét hết các đối tượng là không cần thiết. Vì vậy, ta phải làm cách nào đó để số lượng đối tượng phải duyệt qua giảm đi một lượng đáng kể.
* Để giảm lượng đối tượng được xét, ta phải sử dụng phương pháp phân hoạch không gian. Phương pháp được cho là hiệu quả nhất với một map lớn là QuadTree – phân chia không gian làm bốn phần. QuadTree sẽ chia vùng không gian ra thành những vùng bằng nhau, và nếu đối tượng có thể chứa được ở những vùng không gian nhỏ hơn thì không gian con sẽ được chia nhỏ hơn nữa. Việc phân chia sẽ diễn ra cho đến khi không có không gian con nhỏ hơn chứa được đối tượng đó.
* Khi nhắc đến cây, người ta thường nghĩ ngay đến phương pháp đệ quy. Nếu không sử dụng đệ quy có thể sử dụng hàng đợi để duyệt cây, nhưng việc cài đặt khá là phức tạp. Sử dụng đệ quy giúp cho việc lập trình trở nên đơn giản hơn nhiều, mà không làm giảm hiệu suất của chương trình nhiều.

**

Hình 4. Giới thiệu về cách chia node trong Quadtree

* Các bước thực hiện:

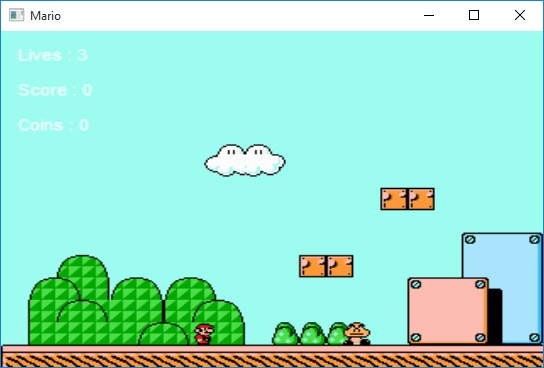
**

Hình 5. Các bước thực hiện việc kiểm tra va chạm trong Quadtree

* Hình chữ nhật xanh va chạm với hình chữ nhật vàng lớn (node mức 1), nhưng không có đối tượng nào ở mức 1.
* Hình chữ nhật xanh va chạm với 2 hình chữ nhật vàng nhỏ ở mức 2 (2 node ở mức 2): cũng không có đối tượng nào, ta tiếp tục kiểm tra các node con.
* Hình chữ nhật xanh va chạm với 4 hình chữ nhật vàng nhỏ ở mức 3 (4 node ở mức 3): có 1 đối tượng, ta đưa đối tượng vào danh sách trả về (danh sách các đối tượng có xảy ra va chạm với hình chữ nhật xanh).
* Hình chữ nhật xanh va chạm với 4 hình chữ nhật vàng nhỏ ở mức 4 (6 node ở mức 4): có 1 đối tượng, ta đưa đối tượng vào danh sách trả về.

1. **Giao diện và cách chơi**
   1. **Giao diện game**

Giao diện game rất đơn giản, chỉ có một màn hình chơi:



Hình 6. Màn hình game

* 1. **Cách chơi**

Game có cách chơi khá đơn giản, sử dụng các phím mũi tên qua lại để di chuyển và 2 phím Z VÀ X với chức năng cụ thể như sau:

* Phím Z: quay đuôi, chạy nhanh (nhấn giữ kết hợp với phím di chuyển), ôm rùa (nhấn giữ và tiến lại sát mai rùa).
* Phím X: nhảy, quẩy đuôi để rơi chậm (khi mario ở dạng có đuôi).

# Nhận xét, kết luận và hướng phát triển

## Nhận xét

Các chức năng trong game đa phần hoạt động tốt. Mario có thể di chuyển, chơi bình thường, quái trong game có thể di chuyển bình thường. Tuy nhiên vì lý do xử lý các hướng va chạm chưa được tốt và quá trình build QuadTree chưa được tối ưu nên dẫn đến va chạm trong game còn chưa ổn định và chưa thực sự chính xác. Ngoài ra, các item trong game như nấm mạng, đạn, chiếc lá, khối chữ P, viên gạch… vẫn hoạt động tốt.

## Kết luận

So với yêu cầu đồ án đặt ra thì game của nhóm đã hoàn thành tương đối các yêu cầu. Mặc dù nhóm đã có rất nhiều cố gắng và bỏ nhiều công sức nghiên cứu, tìm hiểu, thiết kế xây dựng cách giải quyết bài toán nhưng vẫn không tránh khỏi những thiếu sót.

## Hướng phát triển

Trong thời gian tới, nhóm sẽ tiếp tục xây dựng hoàn thiện đồ án, xử lý va chạm tối ưu hơn và làm thêm những cảnh tiếp theo. Nhóm xin chân thành cảm ơn sự đóng góp ý kiến của thầy cô, bạn bè trong suốt quá trình thực hiện đồ án.