TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

BÀI TẬP LỚN

MÔN: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Role-playing game

Nhóm thực hiện: **nhóm 16**
Sinh viên thực hiện: **Đào Minh Tiến - 20190070**
Nguyễn Văn Linh - 20194093
Đinh Tấn Minh - 20194116
Trần Doãn Hiệp - 20190049
Dương Văn Tuyển - 20194210

Lớp: **Khoa học máy tính 06 – Khoá 64**
Mã lớp: **124159**
Giáo viên hướng dẫn: ThS. **Nguyễn Mạnh Tuấn**

MỤC LỤC

LỜI NÓI ĐẦU

1. Lý do và mục đích chọn đề tài

Đề tài được lựa chọn lần này là về một sản phẩm game sinh tồn được viết với ngôn ngữ lập trình java và những kiến thức đã học trong môn lập trình hướng đối tượng. Nhóm đã chọn đề tài này để cùng nhau nghiên cứu và hoàn thiện vì sản phẩm có sử dụng những nguyên lý cốt lõi đã được dạy. Thông qua dự án lần này, nhóm mong muốn kiến thức về môn học được bổ sung và phát triển. Hơn thế nữa, ngành làm game đang rất được thịnh hành trên thị trường, một game sinh tồn với cách chơi dễ hiểu, đồ họa đẹp mắt có thể thu hút tới nhiều đối tượng sử dụng. Đặc biệt khi áp dụng kiến thức lập trình hướng đối tượng và ngôn ngữ lập trình java thì nhà phát hành có thể tìm hiểu thêm để phát triển game trên nền tảng android khi xu hướng bây giờ của người sử dụng là dần chuyển sang những game điện thoại tốn ít dung lượng và dễ sử dụng để giải tỏa stress.

2. Tổng quan về dự án

Dự án lần này là trò chơi mô tả sống động truyện về một chàng hiệp sĩ vượt qua nhiều địa hình hiểm trở, chiến đấu với những con quái và rồng để đòi lại tự do cho ngôi làng của mình.

3. Phạm vi nghiên cứu và phương pháp thực hiện của nhóm.

Áp dụng những kiến thức được truyền đạt và tìm hiểu, nhóm đã dành thời gian để xây dựng cốt truyện, cách thức chơi game, quá trình của nhân vật trong trò chơi, các đối tượng và thuộc tính cần thiết. Từ đó nhóm đã đặc tả yêu cầu của dự án bằng cách xây dựng biểu đồ use case và phân tích thiết kế dự án thông qua việc xây dựng từng biểu đồ lớp, biểu đồ trình tự và thiết kế chi tiết lớp với UML. Cuối cùng nhóm đã đọc những kiến thức được học trên lớp đồng thời tìm hiểu thêm về lập trình game để cùng nhau hoàn thiện đề tài lần này.

4. Kết cấu báo cáo

Báo cáo gồm 4 phần chính được thực hiện tương tự như cách thực hiện dự án. Bước khởi đầu là đặc tả yêu cầu dự án, sau đó đến thiết kế, phân tích dự án. Cuối cùng là nhóm đề ra những công nghệ, thuật toán cần thiết để sử dụng và làm hoàn thiện sản phẩm.

PHÂN CÔNG THÀNH VIÊN TRONG NHÓM

Họ và tên	Email	Điện thoại	Công việc thực hiện	Đánh giá
Trần Doãn Hiệp			Xây dựng UML, use case, lớp KeyAction, Animation, player, gameState, lớp Npc, monster, làm báo cáo, hoàn thiện code, kiểm tra code.	96%
Dương Văn Tuyển			Xây dựng UML, background game (bao gồm Tile, package nature, package world), khung chương trình, báo cáo, entity, creature, hoàn thiện game.	96%
Đào Minh Tiến			Xây dựng background game background game (bao gồm Tile, package nature, package world), music, progress bar, setting game, báo cáo.	92%
Nguyễn Văn Linh			Xây dựng UML, lớp player, entity, creature, boss, video, xử lý tràn viền, dựng hoạt ảnh kết thúc game, làm slide, tổng hợp báo cáo, …	95%
Đinh Tấn Minh			Trưởng nhóm, xây dựng UML, lớp Boss, monster, làm báo cáo, player, creature, kiểm tra, hoàn thiện code, điều phối và lên lịch công việc cho nhóm, hỗ trợ github cho nhóm.	95%

CHƯƠNG 1. KHẢO SÁT, ĐẶC TẢ YÊU CẦU BÀI TOÁN

1.1. Mô tả yêu cầu bài toán

Người chơi điều khiển một hoặc nhiều nhân vật trong một bản đồ được lưu trong một cấu trúc dữ liệu, ví dụ: mảng hai chiều như hình bên, trong đó mỗi ô tương ứng với một dạng bản đồ khác nhau (đất, cỏ, nước…) Trên bản đồ có các quái vật có thể di chuyển được.

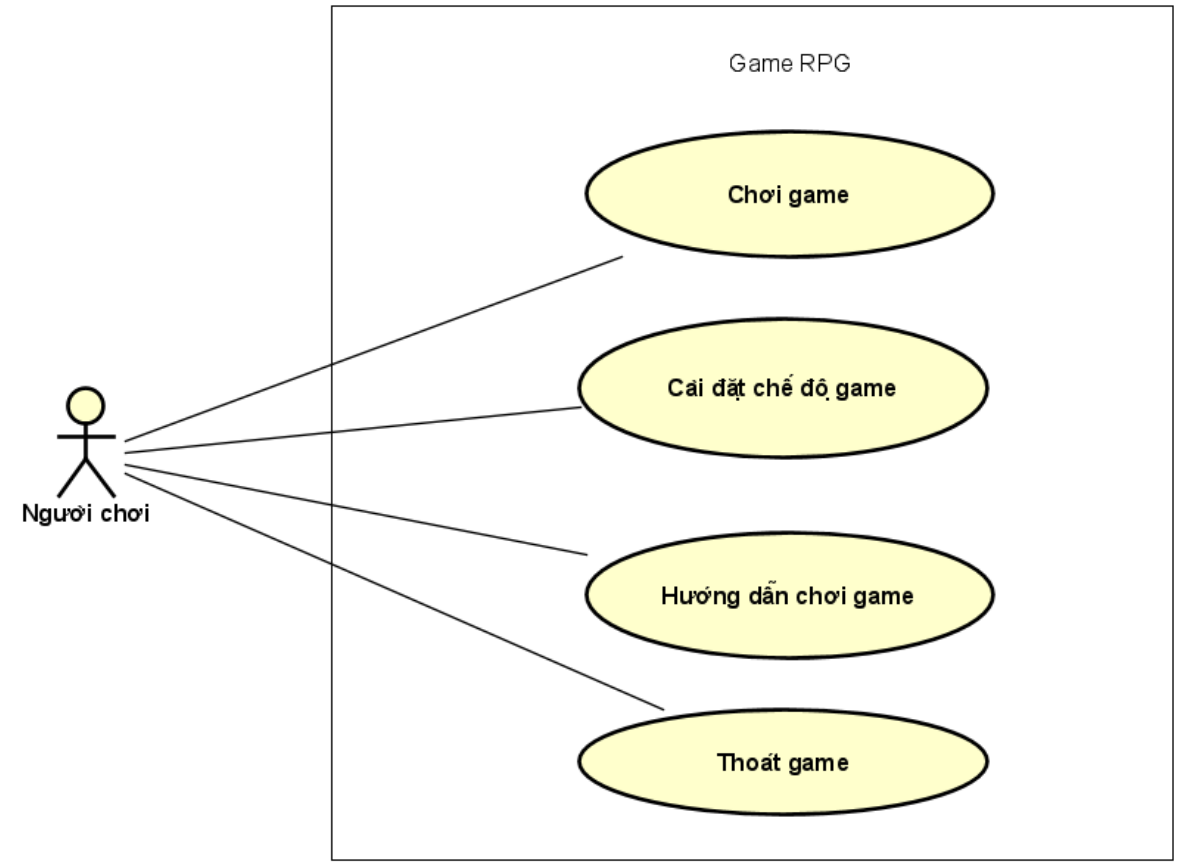
Các nhân vật người chơi điều khiển và quái vật có các chỉ số xác định tình trạng và thể lực (ví dụ HP, MP, Attack, Defense, Speed…). Người chơi có thể tấn công quái vật và sử dụng các kỹ năng đặc biệt. Tương tự, quái vật cũng có thể tìm đến và tấn công người chơi.

Người chơi có thể di chuyển qua lại giữa các bản đồ khác nhau (ví dụ khi đi vào vùng M0, M1, M2… trên bản đồ) hoặc đi đến kết thúc của trò chơi (ví dụ khi đi vào vùng END trên bản đồ).

M1	*1*	*1*	*1*	*2*	*2*	*0*	*0*	*0*	***END***
1	*1*	*1*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
1	*3*	*3*	*3*	*0*	*0*	*0*	*0*	*0*	*5*
1	*3*	*3*	*3*	*3*	*0*	*0*	*0*	*5*	*5*
1	*3*	*3*	*3*	*3*	*0*	*0*	*0*	*0*	*0*
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
0	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*1*	*0*
0	*0*	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*1*
M0	*0*	*1*	*1*	*1*	*1*	*0*	*4*	*0*	*1*
1	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*	*1*

1.2. Biểu đồ use case

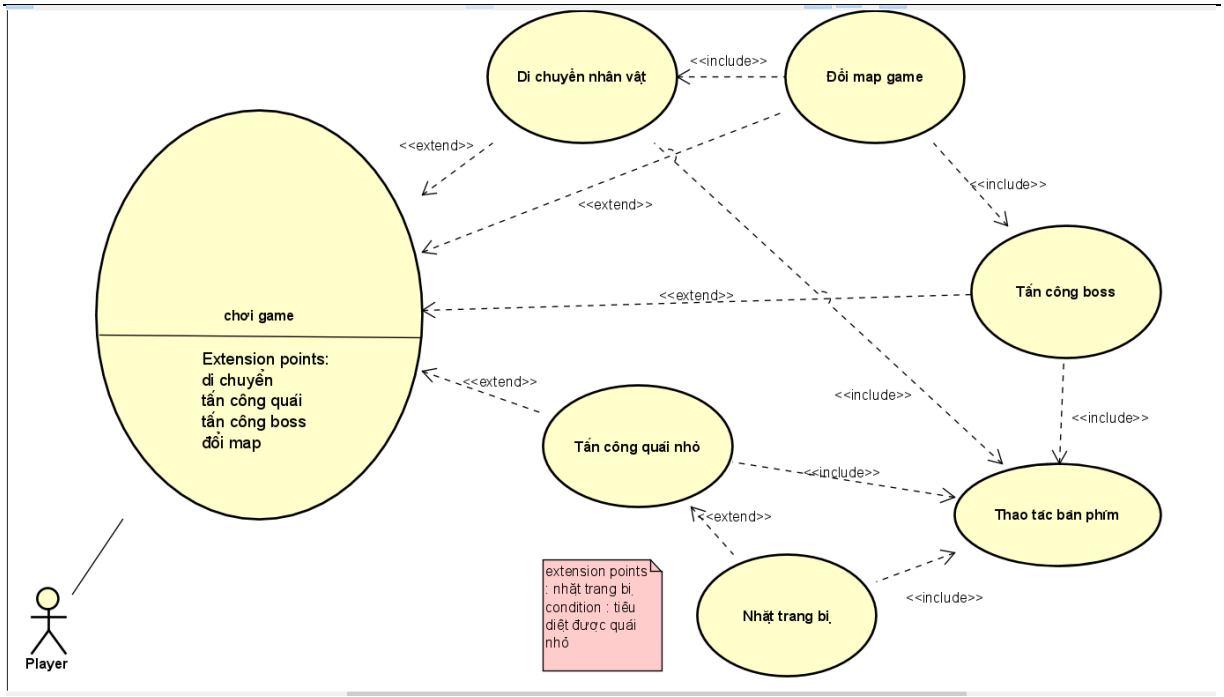
1.2.1. Biểu đồ use case tổng quan



Người chơi khi mở game lên sẽ tương tác với game qua 4 chức năng trên trong đó chức năng chơi game đóng vai trò quan trọng nhất

1.2.2. Biểu đồ use case phân rã mức 2

Biểu đồ use case phân rã cho use case Chơi game



Để chơi game, người chơi có thể điều khiển nhân vật bằng cách di chuyển, tấn công quái nhỏ, boss . Các thao tác trên đều cần sử dụng thao tác với bàn phím. Để di chuyển sang map game mới người chơi cần di chuyển nhân vật tới nơi dịch chuyển và tiêu diệt được boss . Việc nhặt trang bị xuất hiện người các con quái nhỏ bị tiêu diệt

1.3. Đặc tả use case

- Đặc tả use case chơi game.

- Description: Người chơi di chuyển, tấn công quái vật để qua được màn chơi.
- Actor: Player game.
- Trigger: Ấn nút start game
- Pre-Conditions: Đã có game trên máy, người chơi có chỉ số máu > 0
- Post-Conditions: kết thúc màn chơi cũ, chuyển sang màn chơi mới.
- Basic flow:
  1. Người chơi ấn start game.
  2. Di chuyển và tấn công quái vật nhỏ
  3. Giết được quái vật nhỏ, xuất hiện trang bị
  4. Nhặt trang bị tăng một trong ba chỉ số: máu, tấn công, phòng thủ.
  5. Di chuyển và tấn công boss
  6. Giết được boss, nhân vật hồi máu
  7. Sang màn tiếp theo.
- Exception flow:
  3a. Không giết được quái vật nhỏ, bị quái vật nhỏ tấn công hết máu
  3a1. Chuyển sang game over.
  3a2. ấn phím để chơi lại, tiếp tục use case 1.

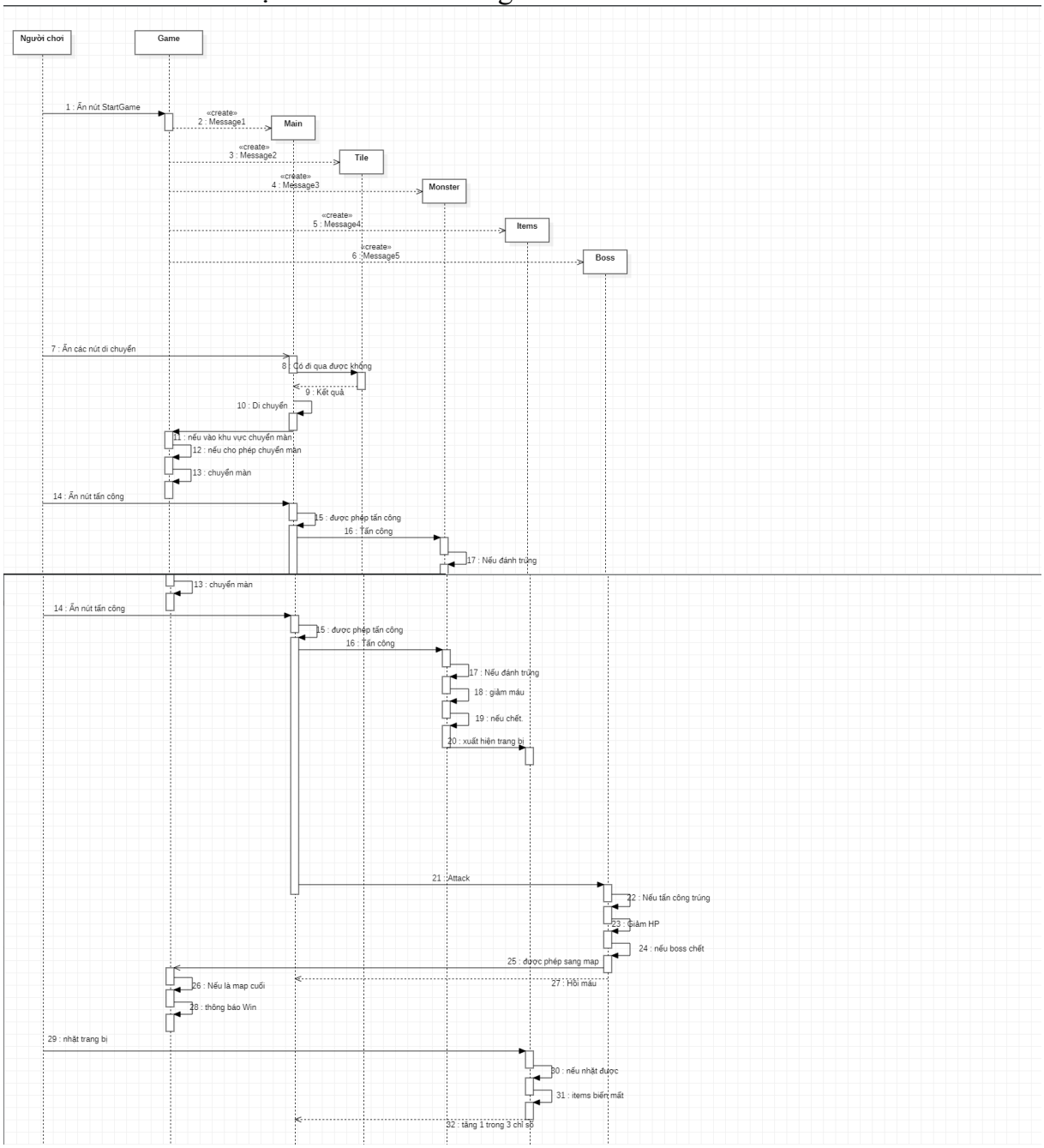
  6a Không giết được boss, bị boss tấn công hết máu
  6a.1 chuyển sang game over.
  6a2. ấn phím để chơi lại, tiếp tục use case 1.

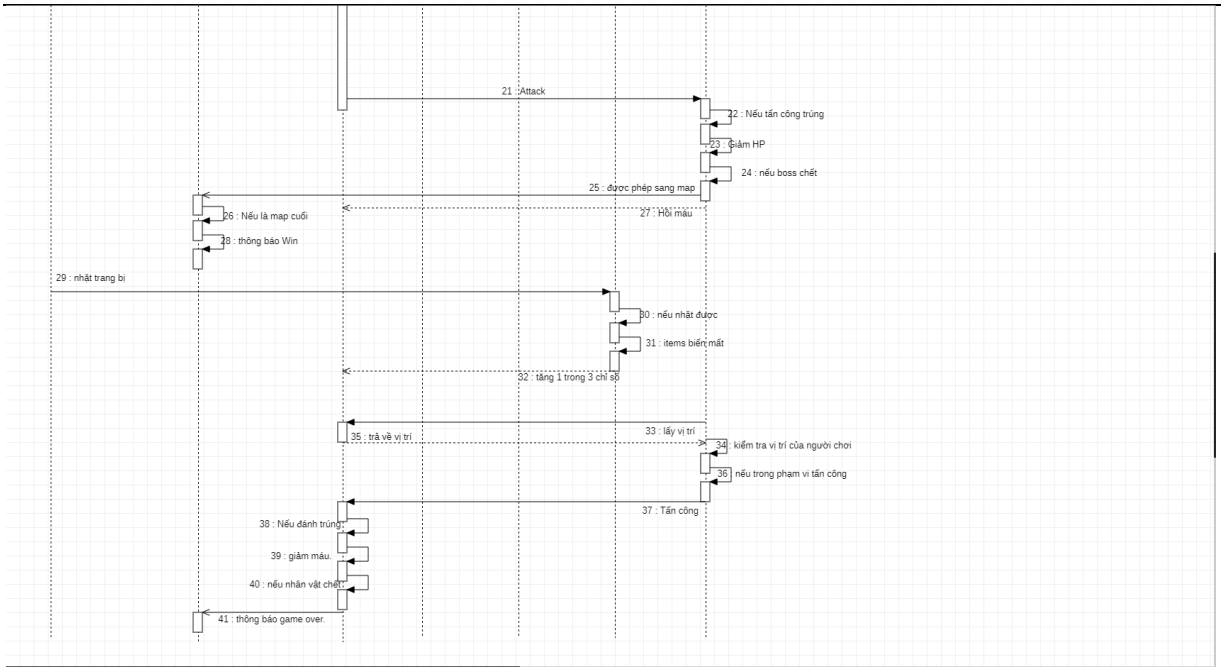
CHƯƠNG 2. PHÂN TÍCH THIẾT KẾ BÀI TOÁN

2.1. Thiết kế Cơ sở dữ liệu hoặc Cấu trúc tệp dữ liệu

2.2. Biểu đồ trình tự

- Biểu đồ trình tự cho use case chơi game





2.3. Biểu đồ gói và biểu đồ lớp

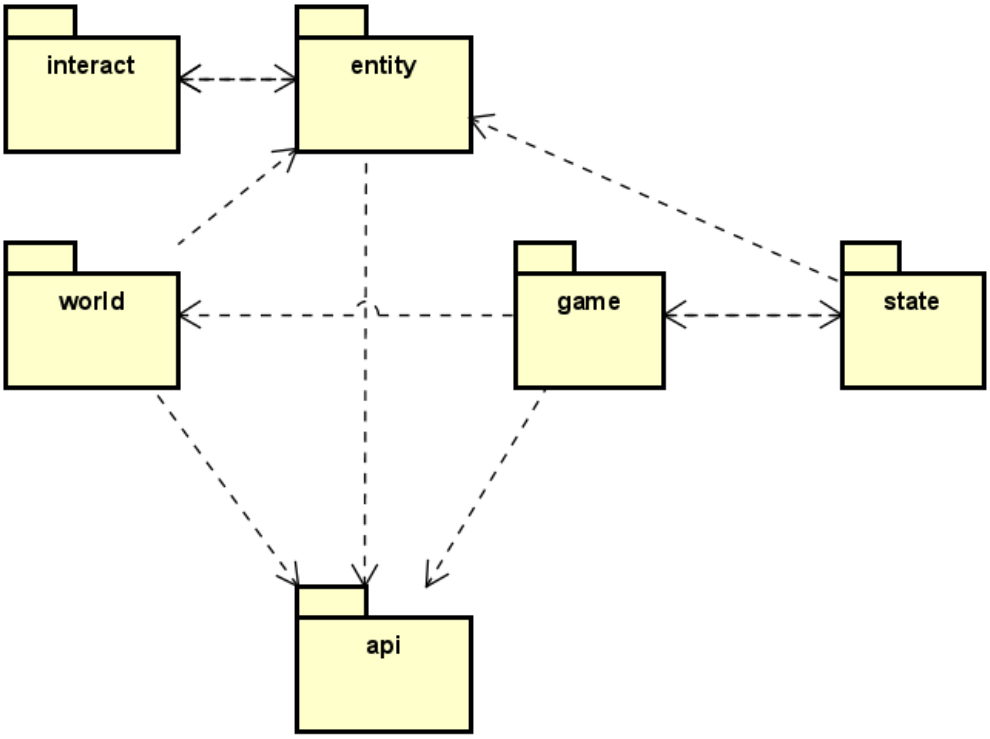
2.3.1. Biểu đồ gói

Chương trình gồm có 6 package lớn:

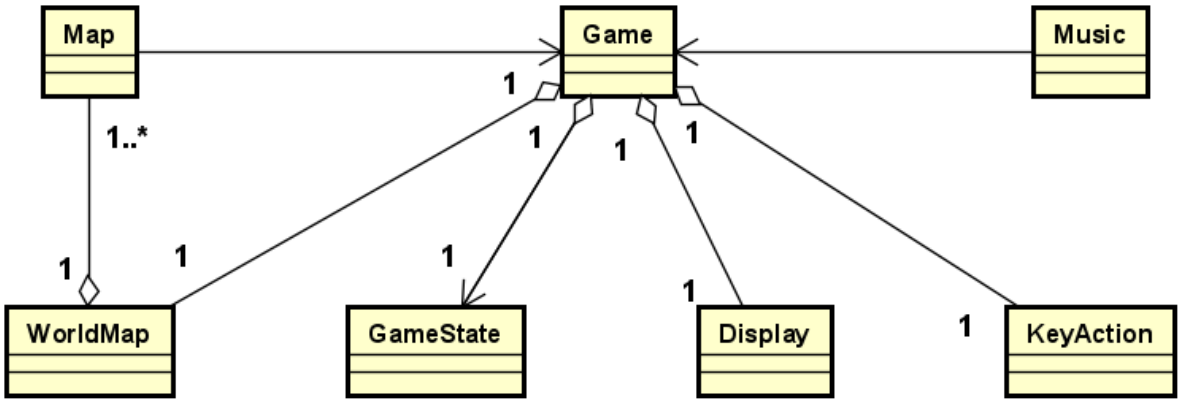
- Package api
- Package entity
- Package game
- Package state
- Package world
- Package interact

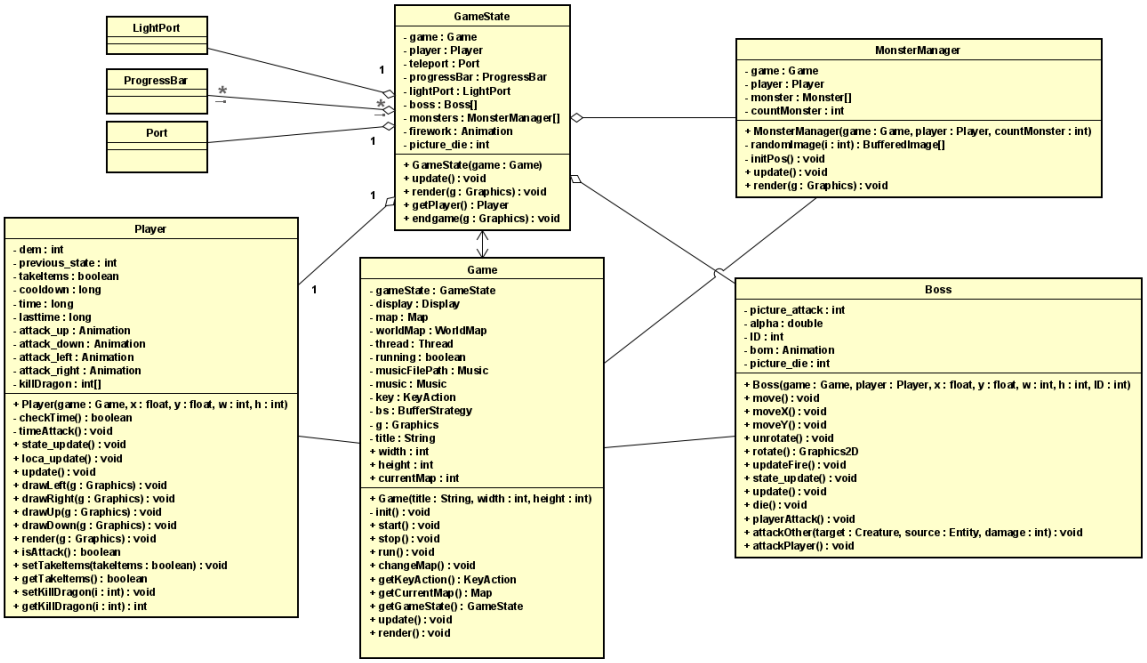
Vai trò của từng package như sau:

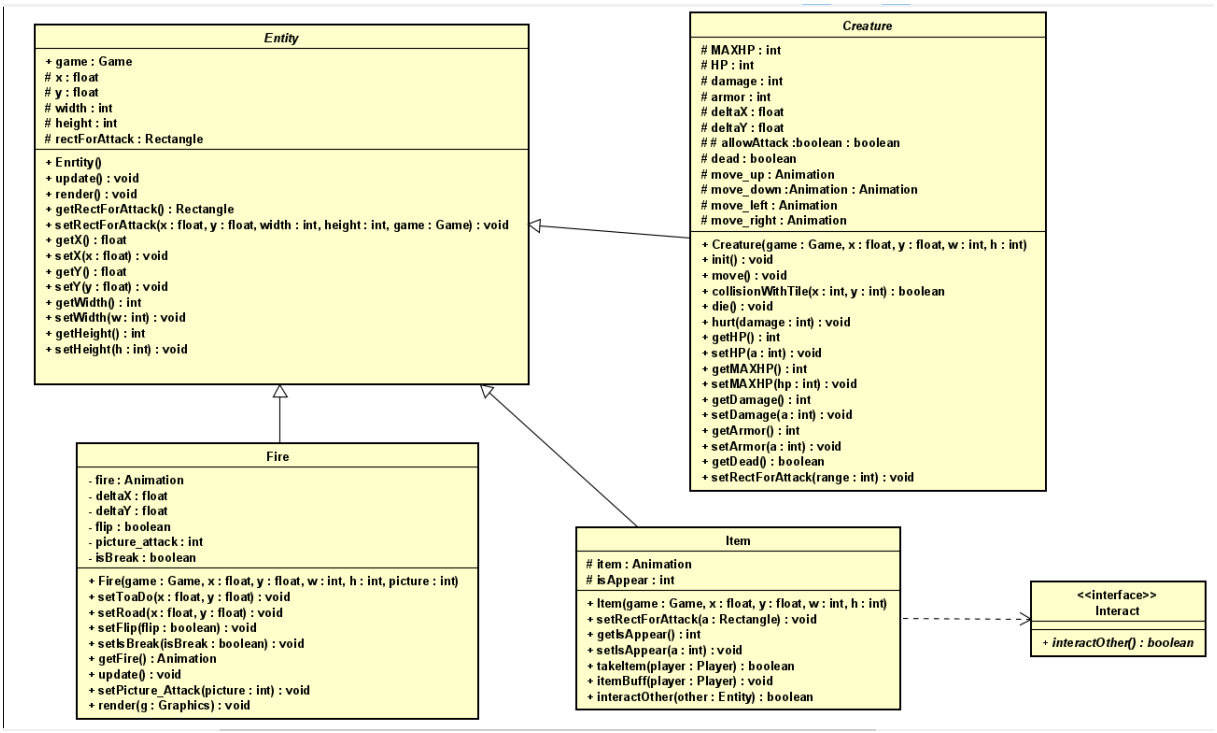
- Package api chứa các lớp KeyAction (xử lý sự kiện bàn phím),Texture(xử lý ảnh),Animation (xử lý hiệu ứng chuyển động)
- Package entity chứa các lớp và package nhỏ hơn liên quan đến các đối tượng trong game như các đối tượng chuyển động như nhân vật ,quái vật nhỏ, boss rồng (package creature) và các trang bị rơi ra khi giết được quái vật (package item)
- Package game chứa các lớp nền tảng của trò chơi như lớp Game (khởi tạo luồng và các đối tượng trong trò chơi , Display (tạo khung giao diện trò chơi) , lớp Laucher (lớp chứa phương thức main) và một số lớp khác có vai trò mang tính bổ trợ như Music , GameStart .
- Package state chứa lớp GameState là lớp khởi tạo và cập nhật các đối tượng chuyển động trong game
- Package world chứa hai lớp Map,WorldMap liên quan đến xử lý bản đồ trong game và các đối tượng tĩnh bên trong nó (cây cối , đất ,nước, …)
- Package interact chứa 2 interface dùng cho xử lý va chạm và tấn công giữa các đối tượng chuyển động trong game.
  Biểu đồ gói

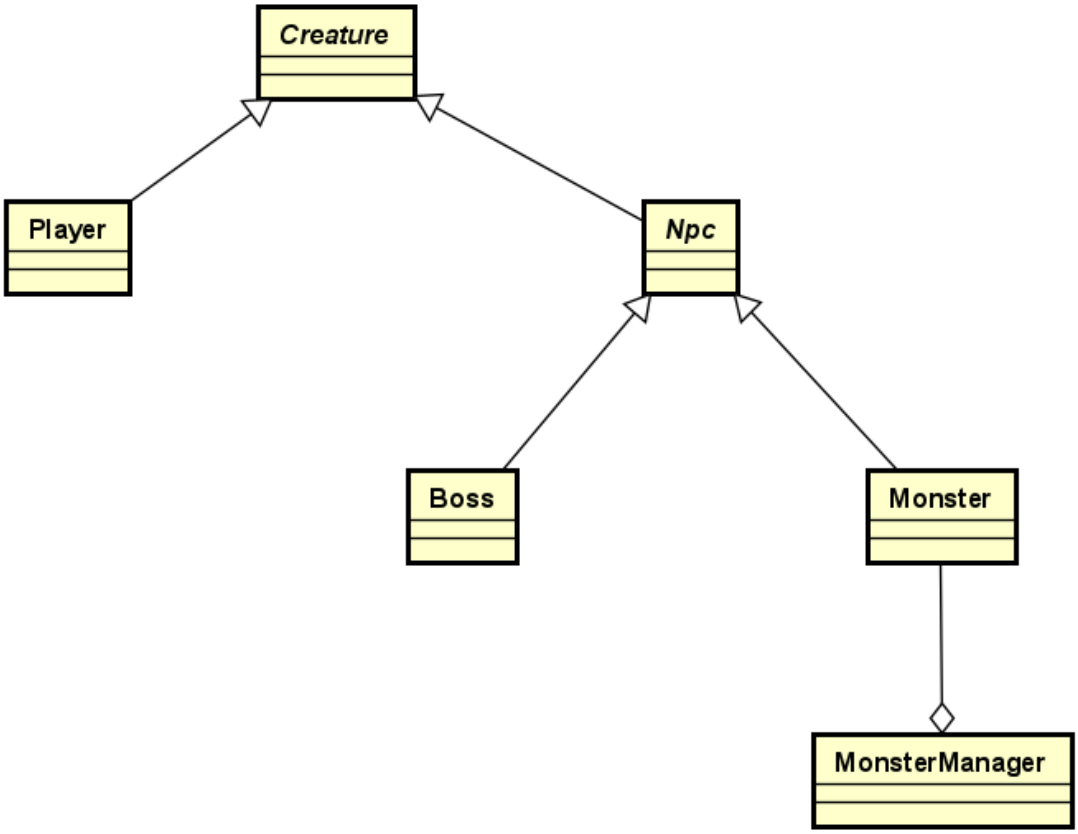

interact
entity
world
game
state
api

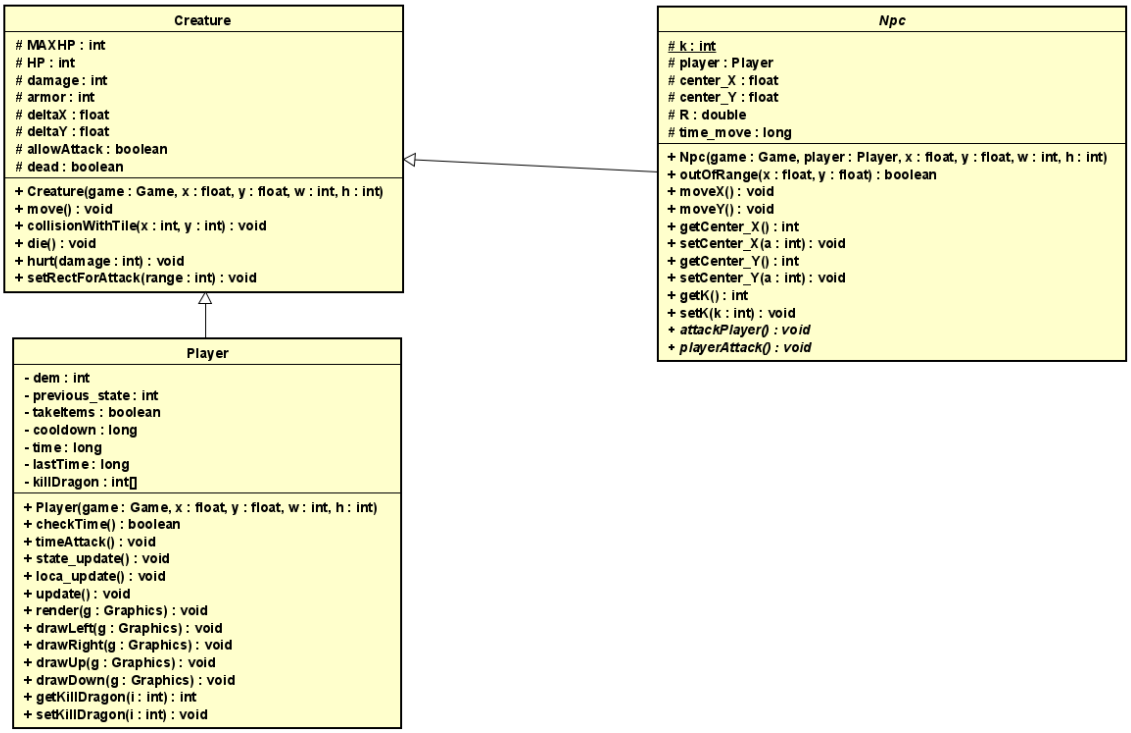
2.3.2. Biểu đồ lớp

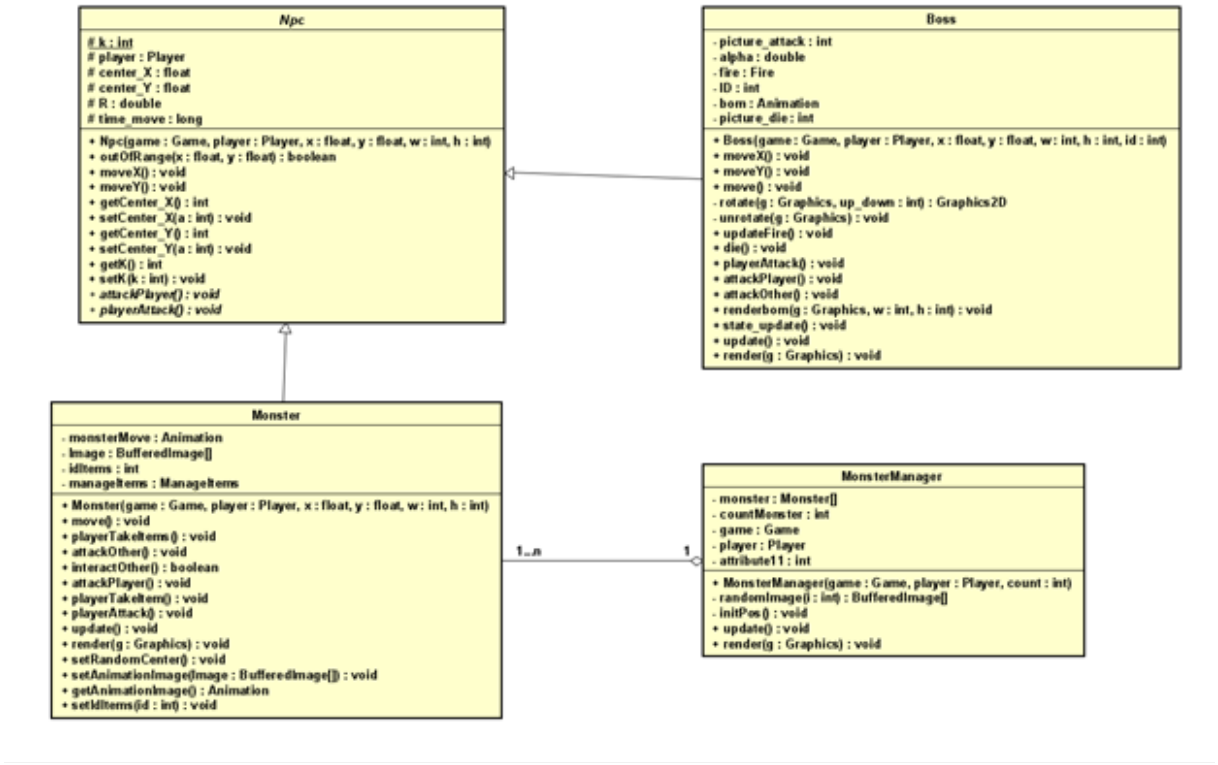


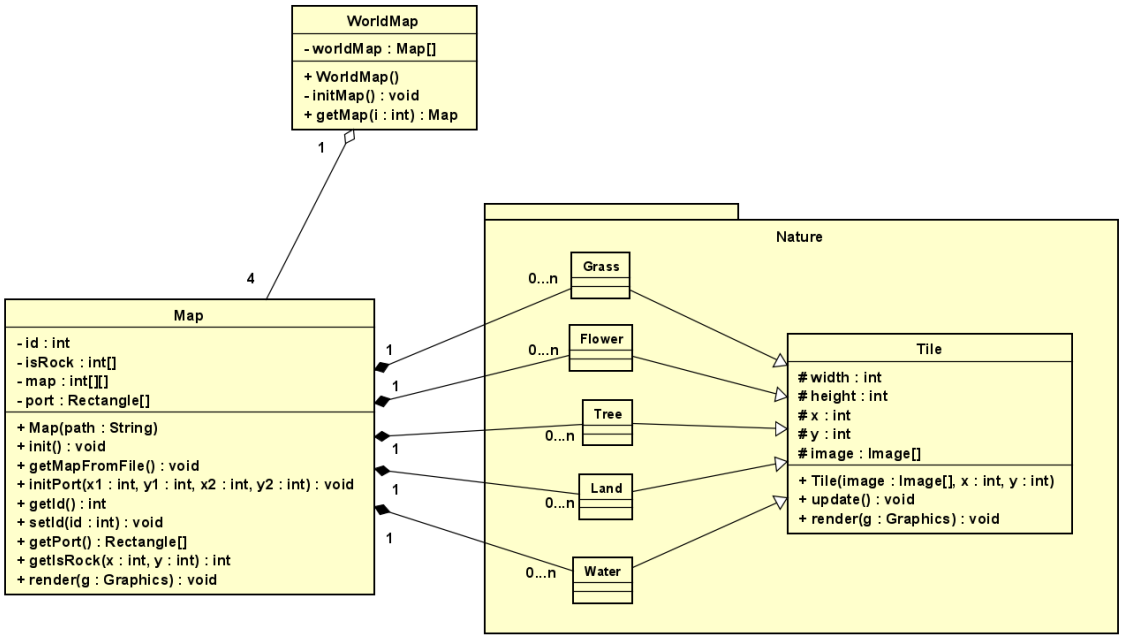


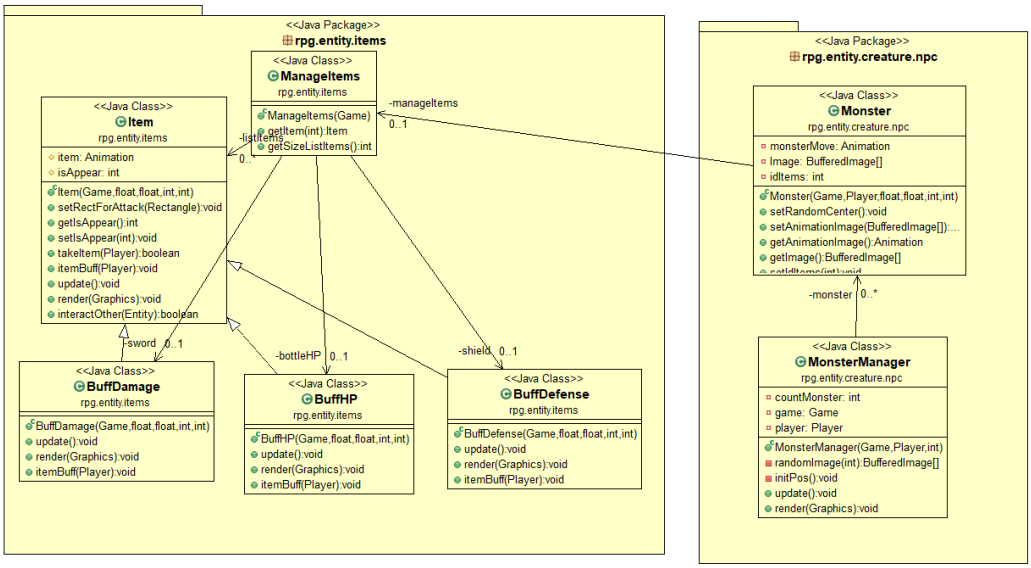

Entity
+ game : Game
# x : float
# y : float
# width : int
# height : int
# rectForAttack : Rectangle
+ Enrtity()
+ update() : void
+ render() : void
+ getRectForAttack() : Rectangle
+ setRectForAttack(x : float, y : float, width : int, height : int, game : Game) : void
+ getX() : float
+ setX(x : float) : void
+ getY() : float
+ setY(y : float) : void
+ getWidth() : int
+ setWidth(w: int) : void
+ getHeight() : int
+ setHeight(h : int) : void
Creature
MAXHP : int
HP : int
damage : int
armor : int
deltaX : float
deltaY : float
allowAttack :boolean : boolean
dead : boolean
move_up : Animation
move_down :Animation : Animation
move_left : Animation
move_right : Animation
+ Creature(game : Game, x : float, y : float, w : int, h : int)
+ init() : void
+ move() : void
+ collisionWithTile(x : int, y : int) : boolean
+ die() : void
+ hurt(damage : int) : void
+ getHP() : int
+ setHP(a : int) : void
+ getMAXHP() : int
+ setMAXHP(hp : int) : void
+ getDamage() : int
+ setDamage(a : int) : void
+ getArmor() : int
+ setArmor(a : int) : void
+ getDead() : boolean
+ setRectForAttack(range : int) : void
Fire
- fire : Animation
- deltaX : float
- deltaY : float
- flip : boolean
- picture_attack : int
- isBreak : boolean
+ Fire(game : Game, x : float, y : float, w : int, h : int, picture : int)
+ setToaDo(x : float, y : float) : void
+ setRoad(x : float, y : float) : void
+ setFlip(flip : boolean) : void
+ setIsBreak(isBreak : boolean) : void
+ getFire() : Animation
+ update() : void
+ setPicture_Attack(picture : int) : void
+ render(g : Graphics) : void
Item
item : Animation
isAppear : int
+ Item(game : Game, x : float, y : float, w : int, h : int)
+ setRectForAttack(a : Rectangle) : void
+ getIsAppear() : int
+ setIsAppear(a : int) : void
+ takeItem(player : Player) : boolean
+ itemBuff(player : Player) : void
+ interactOther(other : Entity) : boolean
<<interface>>
Interact
+ interactOther() : boolean


Creature
Player
Npc
Boss
Monster
MonsterManager


Creature
MAXHP : int
HP : int
damage : int
armor : int
deltaX : float
deltaY : float
allowAttack : boolean
dead : boolean
+ Creature(game : Game, x : float, y : float, w : int, h : int)
+ move() : void
+ collisionWithTile(x : int, y : int) : void
+ die() : void
+ hurt(damage : int) : void
+ setRectForAttack(range : int) : void
Npc
k : int
player : Player
center_X : float
center_Y : float
R : double
time_move : long
+ Npc(game : Game, player : Player, x : float, y : float, w : int, h : int)
+ outOfRange(x : float, y : float) : boolean
+ moveX() : void
+ moveY() : void
+ getCenter_X() : int
+ setCenter_X(a : int) : void
+ getCenter_Y() : int
+ setCenter_Y(a : int) : void
+ getK() : int
+ setK(k : int) : void
+ attackPlayer() : void
+ playerAttack() : void
Player
- dem : int
- previous_state : int
- takeItems : boolean
- cooldown : long
- time : long
- lastTime : long
- killDragon : int[]
+ Player(game : Game, x : float, y : float, w : int, h : int)
+ checkTime() : boolean
+ timeAttack() : void
+ state_update() : void
+ loca_update() : void
+ update() : void
+ render(g : Graphics) : void
+ drawLeft(g : Graphics) : void
+ drawRight(g : Graphics) : void
+ drawUp(g : Graphics) : void
+ drawDown(g : Graphics) : void
+ getKillDragon(i : int) : int
+ setKillDragon(i : int) : void


Npc
k : int
player : Player
center_X : float
center_Y : float
R : double
time_move : long
+ Npc(game : Game, player : Player, x : float, y : float, w : int, h : int)
+ outOfRange(x : float, y : float) : boolean
+ moveX() : void
+ moveY() : void
+ getCenter_X() : int
+ setCenter_X(a : int) : void
+ getCenter_Y() : int
+ setCenter_Y(a : int) : void
+ getK() : int
+ setK(k : int) : void
+ attackPlayer() : void
+ playerAttack() : void
Boss
- picture_attack : int
- alpha : double
- fire : Fire
- ID : int
- bom : Animation
- picture_die : int
+ Boss(game : Game, player : Player, x : float, y : float, w : int, h : int, id : int)
+ moveX() : void
+ moveY() : void
+ move() : void
- rotate(g : Graphics, up_down : int) : Graphics2D
- unrotate(g : Graphics) : void
+ updateFire() : void
+ die() : void
+ playerAttack() : void
+ attackPlayer() : void
+ attackOther() : void
+ renderbom(g : Graphics, w : int, h : int) : void
+ state_update() : void
+ update() : void
+ render(g : Graphics) : void
Monster
- monsterMove : Animation
- image : BufferedImage[]
- idItems : int
- manageItems : ManageItems
+ Monster(game : Game, player : Player, x : float, y : float, w : int, h : int)
+ move() : void
+ playerTakeItems() : void
+ attackOther() : void
+ interactOther() : boolean
+ attackPlayer() : void
+ playerTakeItem() : void
+ playerAttack() : void
+ update() : void
+ render(g : Graphics) : void
+ setRandomCenter() : void
+ setAnimationImage(image : BufferedImage[]) : void
+ getAnimationImage() : Animation
+ setIdItems(id : int) : void
1...n
1
MonsterManager
- monster : Monster[]
- countMonster : int
- game : Game
- player : Player
- attribute11 : int
+ MonsterManager(game : Game, player : Player, count : int)
- randomImage(i : int) : BufferedImage[]
- initPos() : void
+ update() : void
+ render(g : Graphics) : void


WorldMap
- worldMap : Map[]
+ WorldMap()
- initMap() : void
+ getMap(i : int) : Map
1
4
Map
- id : int
- isRock : int[]
- map : int[][]
- port : Rectangle[]
+ Map(path : String)
+ init() : void
+ getMapFromFile() : void
+ initPort(x1 : int, y1 : int, x2 : int, y2 : int) : void
+ getId() : int
+ setId(id : int) : void
+ getPort() : Rectangle[]
+ getIsRock(x : int, y : int) : int
+ render(g : Graphics) : void
Nature
Grass
Flower
Tree
Land
Water
0...n
1
Tile
width : int
height : int
x : int
y : int
image : Image[]
+ Tile(image : Image[], x : int, y : int)
+ update() : void
+ render(g : Graphics) : void


<<Java Package>>
rpg.entity.items
<<Java Class>>
ManageItems
rpg.entity.items
ManageItems(Game)
getItem(int):Item
getSizeListItems():int
-manageItems
0..1
<<Java Class>>
Item
rpg.entity.items
item: Animation
isAppear: int
Item(Game,float,float,int,int)
setRectForAttack(Rectangle):void
getIsAppear():int
setIsAppear(int):void
takeItem(Player):boolean
itemBuff(Player):void
update():void
render(Graphics):void
interactOther(Entity):boolean
-listItems
0..
-sword 0..1
-bottleHP 0..1
-shield 0..1
<<Java Class>>
BuffDamage
rpg.entity.items
BuffDamage(Game,float,float,int,int)
update():void
render(Graphics):void
itemBuff(Player):void
<<Java Class>>
BuffHP
rpg.entity.items
BuffHP(Game,float,float,int,int)
update():void
render(Graphics):void
itemBuff(Player):void
<<Java Class>>
BuffDefense
rpg.entity.items
BuffDefense(Game,float,float,int,int)
update():void
render(Graphics):void
itemBuff(Player):void
<<Java Package>>
rpg.entity.creature.npc
<<Java Class>>
Monster
rpg.entity.creature.npc
monsterMove: Animation
Image: BufferedImage[]
idItems: int
Monster(Game,Player,float,float,int,int)
setRandomCenter():void
setAnimationImage(BufferedImage[])
getAnimationImage():Animation
getImage():BufferedImage[]
-monster 0..*
<<Java Class>>
MonsterManager
rpg.entity.creature.npc
countMonster: int
game: Game
player: Player
MonsterManager(Game,Player,int)
randomImage(int):BufferedImage[]
initPos():void
update():void
render(Graphics):void

2.4. Thiết kế chi tiết lớp

2.4.1. Lớp Game

<table>
<tr><td>Class name: Game</td><td>ID: 1</td><td>Type:</td></tr>
<tr><td colspan="3">Description: Nơi thực hiện các luồng của trò chơi, khởi tạo các thành phần cơ bản của trò chơi</td></tr>
<tr><td colspan="2">Responsibilities
- Khởi tạo, thực thi luồng của trò chơi (Phương thức start)
- Cập nhật trạng thái của các đối tượng trong trò chơi (Phương thức update)
- Biểu diễn các đối tượng, thành phần của trò chơi như Giao diện, Bản đồ game, các đối tượng trong game, ...(render)
- Lấy thông tin về trạng thái của các đối tượng như phím bấm từ bàn phím của người chơi(keyAction), bản đồ game (Map), trạng thái hiện tại của các đối tượng trong game (gameState)</td><td>Collaborators
- Lớp GameState
- Lớp WorldMap, Map
- Lớp KeyAction
- Lớp Display
- Lớp Music</td></tr>
<tr><td colspan="2">Attribites
- gameState (GameState):
- disPlay (Display)
- map (Map)
- worldMap (WorldMap): mảng các Map
- key (KeyAction): thuộc tính để kiểm soát các phím người chơi ấn
- title (String): tiêu đề game
- thread (Thread): luồng của trò chơi
- running (boolean): trả về true nếu game đang chạy
- width, height, currrentMap (int): cho biết kích thước giao diện trò chơi và số hiệu bản đồ hiện tại</td><td>Relationships
- Aggreation: KeyAction, Display, GameState, WorldMap
- Association: Map, Music</td></tr>
</table>

2.4.2. Lớp Entity

<table>
<tr><td colspan="2">Class name: Entity</td><td>ID: 2</td><td>Type:</td></tr>
<tr><td colspan="4">Description: Lớp trừu tượng, mô tả tất cả các đối tượng trong game</td></tr>
<tr><td colspan="2">Responsibilities
- Xác lập, xác định giá trị tọa độ, chiều cao, chiều rộng của đối tượng (các phương thức get/set các thuộc tính x, y, width, height)
- Xác lập, xác định hình bao quanh đối tượng dùng cho việc xử lý va chạm, tấn công giữa các đối tượng (get/setRecForAttack)
-</td><td colspan="2">Collaborators
- Lớp Creature
- Lớp Tile
- Lớp Item
- Lớp Game
- Lớp Fire</td></tr>
<tr><td colspan="2">Attribites
- x, y (float): tọa độ của đối tượng
- width, height (int): bề ngang, chiều cao của đối tượng
- game (Game)
- rectForAttack (Rectangle): hình bao (hình chữ nhật) bao quanh đối tượng</td><td colspan="2">Relationships
- Generlization: Là lớp cha của các lớp Creature, Tile, Fire, Item
- Association: Lớp Game</td></tr>
</table>

2.4.3. Lớp Creature

<table>
<tr><td colspan="2">Class name: Creature</td><td>ID: 2</td><td>Type:</td></tr>
<tr><td colspan="4">Description: Lớp trừu tượng, mô tả các đối tượng sinh vật</td></tr>
<tr><td colspan="2">Responsibilities

- Xác định, xác lập giá trị về chỉ số tấn công, phòng thủ, máu, trạng thái sống chết của đối tượng (get/set các thuộc tính damage, armor, hp, maxHP và get dead)
- Tạo hình bao tấn công của đối tượng (getCollisionBounds)
- Kiểm tra va chạm giữa đối tượng và các vật cản trên bản đồ (phương thức collisionWithTile)
- Giảm máu của đối tượng do bị tấn công (phương thức hurt)</td><td colspan="2">Collaborators

- Lớp Entity
- Lớp Player
- Lớp Animation
- Lớp Attack
- Lớp NPC</td></tr>
<tr><td colspan="2">Attribites

- hp, maxHP(int): chỉ số máu và máu tối đa của đối tượng
- damage, armor (int): chỉ số tấn công, phòng thủ của đối tượng
- deltaX, deltaY (float): độ dịch chuyển vị trí của đối tượng
- allowAttack, dead (boolean): trả về đối tượng có thể tấn công không, đối tượng còn sống không</td><td colspan="2">Relationships

- Generlization: là lớp con của lớp Entity, lớp cha của lớp Player, NPC
- Association: Animation</td></tr>
</table>

2.4.4. Lớp Player

<table>
<tr><td colspan="2">Class name: Player</td><td>ID: 2</td><td>Type:</td></tr>
<tr><td colspan="4">Description: Lớp khởi tạo, cập nhật trạng thái nhân vật</td></tr>
<tr><td colspan="2">Responsibilities

- Cập nhật trạng thái nhân vật (máu, vị trí, thời gian delay đòn tấn công) (phương thức state_update).
- Di chuyển nhân vật (kế thừa từ lớp cha là lớp Creature).
- Biểu diễn hiệu ứng Animation di chuyển, tấn công của nhân vật (phương thức render).
- Lấy, tạo giá trị về thuộc tính giết boss, nhặt trang bị (get/setTakeItems, get/setKillDragon)</td><td colspan="2">Collaborators

- Lớp Animation
- Lớp Texture
- Lớp Creature</td></tr>
<tr><td colspan="2">Attribites
- dem (int): dùng để kiểm tra điều kiện tấn công của nhân vật
- previous_state (int): dùng để lưu trạng thái animation nhân vật
- takeItems (boolean): cho biết nhân vật có thể nhặt item không
- cooldown (long): thời gian delay đòn tấn công của nhân vật
- killDragon (int[]): lưu trữ giá trị cho việc giết boss</td><td colspan="2">Relationships
- Generlization: Lớp con của lớp Creature
- Association: Animation,Game
- Dependency: Texture</td></tr>
</table>

2.4.5. Lớp Npc

<table>
<tr><td>Class name: Npc</td><td>ID: 2</td><td>Type:</td></tr>
<tr><td colspan="3">Description: Lớp trừu tượng, mô tả chung cho quái vật và boss trong game</td></tr>
<tr><td colspan="2">Responsibilities
- Cho phép đối tượng tự di chuyển một cách ngẫu nhiên (phương thức moveX, moveY)
- Xác lập, xác định vị trí ban đầu, bán kính di chuyển, thời gian di chuyểnget/set các thuộc tính center_X, center_Y, radius, time_move)</td><td>Collaborators
- Lớp Monster
- Lớp Boss
- Lớp Creature
- Lớp Player</td></tr>
<tr><td colspan="2">Attribites
- R, k (int): bán kính di chuyển và thông số để tang máu của đối quái vật tùy theo chế độ game
- center_X, center_ Y (float): vị trí ban đầu của đối tượng và cũng là tâm của vòng tròn di chuyển.
- time_move(long): thời gian di chuyển của đối tượng</td><td>Relationships
- Generlization: là lớp cha của lớp Monster, Boss; là lớp con của lớp Creature
- Association: lớp Player</td></tr>
</table>

2.4.6. Lớp GameState

<table>
<tr><td colspan="2">Class name: GameState</td><td>ID: 3</td><td>Type:</td></tr>
<tr><td colspan="4">Description: Lớp khởi tạo, cập nhật và Biểu diễn các đối tượng nhân vật, quái nhỏ và boss</td></tr>
<tr><td colspan="2">Responsibilities
- Khởi tạo đối tượng game, nhân vật, boss, danh sách quái nhỏ, cổng dịch chuyển (teleport, lightport) (phương thức khởi tạo)
- Cập nhật trạng thái của các đối tượng trên (phương thức update)
- Biểu diễn các đối tượng lên màn hình (phương thức render)
- Lấy giá trị thuộc tính player và lightPort (phương thức getPlayer, getLightPort)
- Vẽ hiệu ứng Animation lúc win game</td><td colspan="2">Collaborators
- Lớp Game
- Lớp Player
- Lớp Boss
- Lớp MonsterManager
- Lớp Port
- Lớp LightPort
- Lớp Animation
- Lớp ProcessBar</td></tr>
<tr><td colspan="2">Attribites
- game (Game)
- player (Player)
- boss (Boss[]): mảng các boss ứng với các bản đồ
- monsters(MonsterManager[]): mảng các tập hợp quái nhỏ
- teleport (Port): cổng dịch chuyển
- lightPort (LightPort): hiệu ứng ánh sáng của cổng dịch chuyển
- firework (Animation): hiệu ứng pháo hoa
- processBar(ProcessBar): thanh trạng thái của nhân vật ở góc màn hình</td><td colspan="2">Relationships
- Association: Lớp Game
- Aggreation: Lớp Player, lớp Boss, lớp MonsterManager, lớp Port, lớp LightPort, Animation, ProcessBar</td></tr>
</table>

2.4.7. Lớp Map

Class name: Map	**ID:** 4	**Type:**
Description: Lớp khởi tạo và biểu diễn các đối tượng như cây, cỏ, đất, nước, nhà cửa, ... lên màn hình		
Responsibilities - Đọc Map từ file là một ma trận (phương thức getMapFromFile) - Khởi tạo và lấy giá trị về vị trí cổng dịch chuyển của Map (phương thức initPort, getPort) - Kiểm tra xem nhân vật có thể di chuyển qua một ô trong map không (phương thức getIsRock) - Biểu diễn các đối tượng lên màn hình (phương thức render)		**Collaborators** - Các lớp như Lake, Land, Tree, Grass, … (gọi chung là các lớp Nature).
Attribites - map (int[][]): mảng 2 chiều để lưu giá trị đọc được từ file - port (Rectangle []): mảng hình bao của các cổng dịch chuyển - id (int): id ứng với mỗi bản đồ riêng dùng để xử lý cho việc chuyển đổi giữa các bản đồ - isRock[]: lưu trư giá trị của các vùng đi qua được hay không đi qua được trên bản đồ (bản đồ được tạo từ các ô 32*32 , mỗi vùng là một ô)		**Relationships** - Generlization: là lớp con của lớp Entity, lớp cha của lớp Player, NPC - Association: Animation

2.4.8. Phân tích các kỹ thuật thiết kế và lập trình hướng đối tượng

Trong bài tập lớn này, nhóm chúng em đã áp dụng cả 4 đặc tính cơ bản của Lập trình hướng đối tượng. Đó là:

- **Tính trừu tượng (Abstraction)**
- **Tính đóng gói (Encapsulation)**
- **Tính kế thừa (Inheritance)**
- **Tính đa hình (Polymorphism)**

Cụ thể, trong nội dung bài tập lớn này, các đặc tính trên đã được áp dụng như sau:

- **Tính trừu tượng (Abstraction)**
  Tính trừu tượng được thể hiện rõ nhất ở các lớp Entity, Creature, Player, Boss thông qua các thuộc tính của lớp.
  Đối với lớp Entity là lớp đại diện cho tất cả các sự vật trong game nên được xây dựng là một lớp trừu tượng và có các thuộc tính chung cho tất cả là tọa độ của đối tượng (x, y), kích thước của đối tượng (width, height) và hình bao cho đối tượng (rectForAttack).

```
public abstract class Entity {
    protected float x, y;
    protected int width, height;
    public Game game;
    // Minh sua //
    protected Rectangle rectForAttack;
```

  Ở lớp Creature là lớp đại diện cho các sinh vật trong game thì ngoài các thuộc tính kế thừa từ lớp cha (lớp Entity) thì còn có thêm một số thuộc tính là điểm chung của nhóm đối tượng này như hp, maxHP (máu và máu tối đa của đối tượng), damage, armor (chỉ số tấn công và phòng thủ của đối tượng), dead (cho biết đối tượng còn sống hay đã chết), allowAttack (cho biết đối tượng có thể tấn công được hay không), …

```
public abstract class Creature extends Entity {
    protected int MAXHP;
    protected int HP;
    protected int damage, armor;
    protected float deltaX, deltaY;
    protected boolean allowAttack,dead;
    protected Attack attackOther;
    // protected Attack attack;
```

- **Tính đóng gói (Encapsulation)**
  Tính đóng gói được thể hiện thông qua việc sử dụng chỉ định truy cập của các thuộc tính. Đối với các thuộc tính, phương thức được kế thừa từ lớp cha xuống lớp con thì chỉ định truy cập được sử dụng là protected, những thuộc tính mà chỉ có lớp đó sử dụng thì được giới hạn phạm vi qua chỉ định private. Ở lớp Entity các thuộc tính tọa độ (x, y), kích thước (width, height) và hình bao(rectForAttack) đều được các lớp con kế thừa nên có chỉ định truy cập là protected. Ở lớp Player các thuộc tính như cooldown, time, … chỉ được sử dụng ở trong lớp này nên có chỉ định private. Các đối tượng khác không thể tác đông trực tiếp đến dữ liệu được che giấu bên trong các lớp mà buộc phải thông qua các phương thức công khai do đối tượng cung cấp (getter & setter) (ví dụ ở lớp Player có phương thức công khai get/setKillDragon để lấy hoặc thay đổi giá trị thuộc tính killDragon

```
public abstract class Entity {
    protected float x, y;
    protected int width, height;
    public Game game;
    // Minh sua //
    protected Rectangle rectForAttack;
```

```
public class Player extends Creature {
    private int dem = 0;
    private int previous_state = 0;
    private boolean takeItems;
    private long cooldown = 1000, time = 5000, lasttime = 0;
    private Animation attack_up, attack_down, attack_left, at
    private int killDragon[];
```

- **Tính kế thừa (Inheritance)**

- Tránh việc phải sử dụng lại code nhiều lần, tận dụng lại những phương thức, thuộc tính và các đoạn code sẵn có thì tính kế thừa đã được sử dụng trong bài tập lớn. Cụ thể:
  + Lớp Entity là lớp đại diện cho các sự vật trong game do đó nó bao gồm các thuộc tính cơ bản như tọa độ (x, y) để xác định vị trí trên bản đồ game, thông số về kích thước (width, height), cũng như các hình bao (rectForAttack) được dùng để phát hiện tương tác với các sự vật khác trong game.
  + Do đó các lớp như Creature (đại diện cho các sinh vật sống, Item (đại diện cho trang bị của người chơi), Fire (đại diện cho quả cầu lửa được sử dụng bởi boss khi tấn công người chơi) đều kế thừa từ lớp Entity.
  + Lớp Creature, vì là lớp đại diện các sinh vật sống, nên Creature ngoài các thuộc tính và phương thức kế thừa từ lớp Entity, Creature còn có những thuộc tính chung điển hình như máu, chỉ số tấn công, khả năng phòng thủ. Cùng với các thuộc tính, các sinh vật sống cũng có các khả năng chung như di chuyển, tấn công, chết. Do đó lớp Creature bao gồm các phương thức như move, die, …
  + Lớp NPC, đây là lớp đại diện cho các nhân vật trong game có khả năng tự động di chuyển, tấn công người chơi, và là sinh vật sống. Do đó, lớp Npc thừa kế từ lớp creature và thêm các phương thức như tấn công người chơi, di chuyển trong một phạm vi nhất định để đến lớp cụ thể như lớp Boss, Monster có thể sử dụng mà không cần viết lại.

```
public void move() {
    if (x + width + deltaX >= GameStart.MAX_WIDTH || x + deltaX <= 0)
        return;
    if (y + deltaY <= 0 || y + height + deltaY >= GameStart.MAX_HEIGHT)
        return;
```

(Lớp Creature kế thừa thuộc tính x,y từ lớp cha)

- **Tính đa hình (Polymorphism)**

Tính đa hình được áp dụng trong bài tập lớn này thông qua việc ghi đè phương thức (Method Overriding) và nạp chồng phương thức.
Ở lớp Creature có phương thức move để di chuyển đối tượng

```java
// move, xu ly va cham
public void move() {
    if (x + width + deltaX >= GameStart.MAX_WIDTH || x + deltaX <= 0)
        return;
    if (y + deltaY <= 0 || y + height + deltaY >= GameStart.MAX_HEIGHT)
        return;

    int tx, ty, tyY;
    if (deltaX > 0) {
        tx = (int) (deltaX + rectForAttack.x + rectForAttack.width) / 32;
        ty = (int) (rectForAttack.y) / 32;
        tyY = (int) (rectForAttack.y + rectForAttack.height) / 32;
        if (!collisionWithTile(tx, ty) && !collisionWithTile(tx, tyY)) {
            x += deltaX;
        } else {
            x = tx * 32 + -rectForAttack.width - 1 - rectForAttack.x + x;
        }
    } else if (deltaX < 0) {
        tx = (int) (deltaX + rectForAttack.x) / 32;
        ty = (int) (rectForAttack.y) / 32;
        tyY = (int) (rectForAttack.y + rectForAttack.height) / 32;
        if (!collisionWithTile(tx, ty) && !collisionWithTile(tx, tyY)) {
            x += deltaX;
        } else {
            x = tx * 32 + 32 - rectForAttack.x + x;
        }
    }
```

Lớp cháu là Monster (kế thừa lớp con của lớp Creature là lớp Npc) ghi đè lại phương thức này do các đối tượng Monsters di chuyển một cách tự động và ngẫu nhiên nên cần ghi đè lại phương thức

```
@Override
public void move()
{
    super.move();
    if(System.currentTimeMillis() - time_move > 1000 || !outOfRange(x, y))
    {
        time_move = System.currentTimeMillis();
            // rand = Math.random();
            if (ThreadLocalRandom.current().nextInt(1, 100) < 50) {
                moveX();
            } else {
                moveY();
            }
    }
    // x+= deltaX;
    // y+= deltaY;

}
```

- Ngoài ra về overloading còn thể hiện ở phương thức thiết lập bao cho đối tượng. Cụ thể:

+ Ở lớp Entity có phương thức:

```
public void setRectForAttack( int x, int y, int width, int height)
{
    rectForAttack.setBounds(x, y, width, height);;
}
```

+ Ở lớp Item, lớp thừa kế từ lớp Entity:

```
public void setRectForAttack(Rectangle a) {
    rectForAttack.setBounds(a);
}
```

+ Ở lớp creature, lớp thừa kế từ lớp Entity

```
public void setRectForAttack(int range) {
    this.setRectForAttack((int) (rectForAttack.x - range), (int) (rectForAttack.y - range),
            rectForAttack.width + range * 2, rectForAttack.height + range * 2);
}
```

CHƯƠNG 3. CÔNG NGHỆ VÀ THUẬT TOÁN SỬ DỤNG

3.1. Các công nghệ đã sử dụng trong bài tập lớn

- StarUML để thiết kế UML diagram và use case diagram
- Eclipse: IDE phổ biến hỗ trợ NNLT Java. Eclipse là phần mềm miễn phí với nhiều plugin tiện ích.
- Visual studio code: IDE phổ biến hỗ trợ NNLT Java
- Github: là một hệ thống quản lý dự án và phiên bản code. Github giúp nhóm quản lý, lưu trữ và làm việc với dự án một cách thuận tiện.
- Adobe illustrator: Phần mềm đồ họa dùng để thiết kế hình ảnh của các đối tượng
- Các phần mềm thiết kế map

3.2. Kiến thức đã được sử dụng trong bài tập lớn

- 4 đặc tính cơ bản của Lập trình hướng đối tượng:
  - Tính trừu tượng (Abstraction)
  - Tính đóng gói (Encapsulation)
  - Tính kế thừa (Inheritance)
  - Tính đa hình (Polymorphism)
- Các kiến thức xử lý ảnh trong lập trình game với java
- Và các kiến thức, công nghệ, thuật toán khác liên quan đến bộ môn lập trình hướng đối tượng

3.3. Thuật toán được sử dụng trong bài tập lớn

- Thuật toán 1: Sử dụng hình bao và giao hình bao để xác định tương tác giữa sự vật trong game.
  - Trong java kiểu dữ liệu Rectangle có phương thức là intersects, dùng để xác định xem hai hình chữ nhật có giao nhau hay không.
  - Dựa trên cơ sở đó, nhóm đã thiết kế thuộc tính rectForAttack có kiểu dữ liệu cho lớp Entity. Đây là một hình chữ nhật bao quanh đối tượng, nếu hai hình bao giao nhau điều đó có nghĩa là hai đối tượng đã xảy ra va chạm. Tùy và tình huống cụ thể sẽ dùng các thuật toán để xử lý tương tác.
- Thuật toán 2: Sử dụng thuật toán cho phép các Npc (Non player character) tự do di chuyển.
  - Mục đích: các Npc có thể tự động di chuyển trong các phạm vi một cách ngẫu nhiên.
  - Ý tưởng: các Npc di chuyển trong một phạm vi hình tròn, npc được lựa chọn ngẫu nhiên theo một trong các hướng là lên, xuống, trái, phải. Sau khi lựa chọn hướng Npc sẽ di chuyển trong một khoảng thời gian xác định. Nếu Npc vượt ra ngoài phạm vi hướng sẽ được đảo ngược lại.

- Để các Npc có thể di chuyển ngẫu nhiên, nhóm đã sử dụng các thuộc tính là time_move, center_X, center_Y, R.
- Phạm vi là một hình tròn có bán kính là R. Trong đó center_X, center_Y là trung tâm của hình tròn.
- Time_move: đây là biến thời gian tính theo đơn vị ms. Biến time_move dùng để xác định khoảng thời gian mà Npc di chuyển theo một hướng cố định.

```
public void moveX() {
    deltaX = 0;
    deltaY = 0;
    // if(this.x > 0 && this.x < GameStart.MAX_WIDTH-32) {
    if (outOfRange(this.x, this.y)) {
        if (ThreadLocalRandom.current().nextInt(1, 100) < 50) {
            deltaX = -1.5f;
        } else
            deltaX = 1.5f;
    } else {
        if (center_X > x) {
            deltaX = 1.5f;
        } else
            deltaX = -1.5f;
    }

}
```

- Thuật toán 3: Sử dụng thuật toán cho phép Boss phát hiện và tấn công người chơi khi đi vào phạm vi của quái vật.
  - Mục đích: Boss di chuyển tự do trong phạm vi, không chịu cản trở bởi vật cản, có thể phát hiện, tấn công và di chuyển về phía người chơi nếu người chơi đi vào phạm vi tấn công.
  - Ý tưởng: ở mỗi lần di chuyển kiểm tra vị trí của người chơi xem đã vào phạm vi chưa. Nếu đã vào phạm vi thì Boss chuyển sang chế độ tấn công, và di chuyển về phía người chơi.
  - Đầu tiên, xét vị trí của người chơi. Nếu người chơi trong phạm vi hoặc nếu người chơi ở gần Boss nhưng không trong phạm vi thì Boss tấn công. Nếu người chơi ở xa thì Boss di chuyển về phía người chơi để tăng tỉ lệ sát thương.
  - Tiếp theo xét vị trí tương đối tọa độ X của Boss với người chơi. Nếu xa hơn 50 thì tiến đến gần.
  - Tương tự với vị trí của Y. Boss sẽ ở cách người chơi một đoạn để tránh bị tấn công.

```
    }
    public boolean distance(float x1, float x2, float y1, float y2)
    {
        double a = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
        if (a >= 70) {
            return false;
        } else
            return true;
    }

    public void moveX() {
        if (outOfRange(player.getX(), player.getY()) || distance(x, player.getX(), y, player.getY())) {
            allowAttack = true;

            if (Math.abs(player.getX() - x) >= 50f) {
                if (player.getX() > x + 3.0f) {
                    deltaX = 3.0f;
```

- Thuật toán 4: Boss tấn công người chơi bằng cách phun lửa.
  - Sử dụng đối tượng Fire là cầu lửa, thừa kế từ lớp entity.
  - Fire có điểm xuất phát là từ Boss và điểm dừng là vị trí người chơi tại thời điểm rồng tấn công.
  - Fire di chuyển theo công thức:

```
public void setRoad(float player_x, float player_y) {

    if (player_y > this.y)
        flip = true;
    else
        flip = false;
    deltaX = (player_x - this.x) / fire.getImageLength() * 1.0f;
    deltaY = (player_y - this.y) / fire.getImageLength() * 1.0f;

}
```

- Ngoài ra còn những thuật toán khác như: Thuật toán xử lý mức độ khó và dễ của màn chơi, thuật toán xử lý va chạm của người khi di chuyển và khi tấn công.

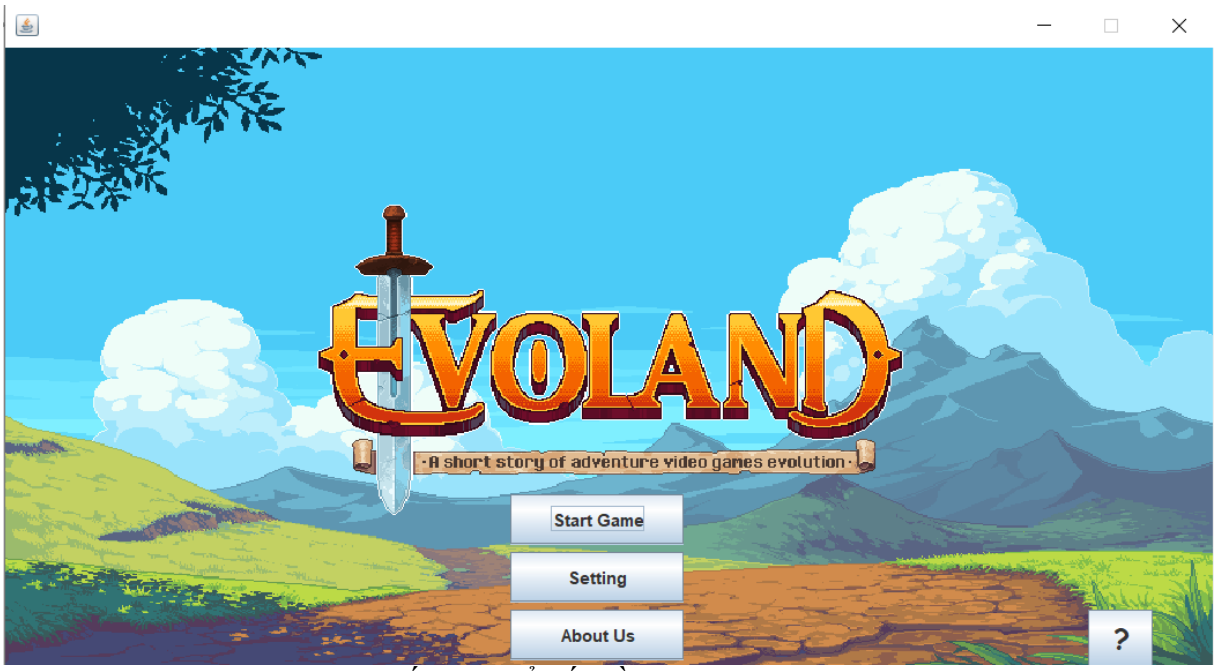
CHƯƠNG 4. XÂY DỰNG CHƯƠNG TRÌNH MINH HỌA

4.1. Kết quả chương trình minh họa

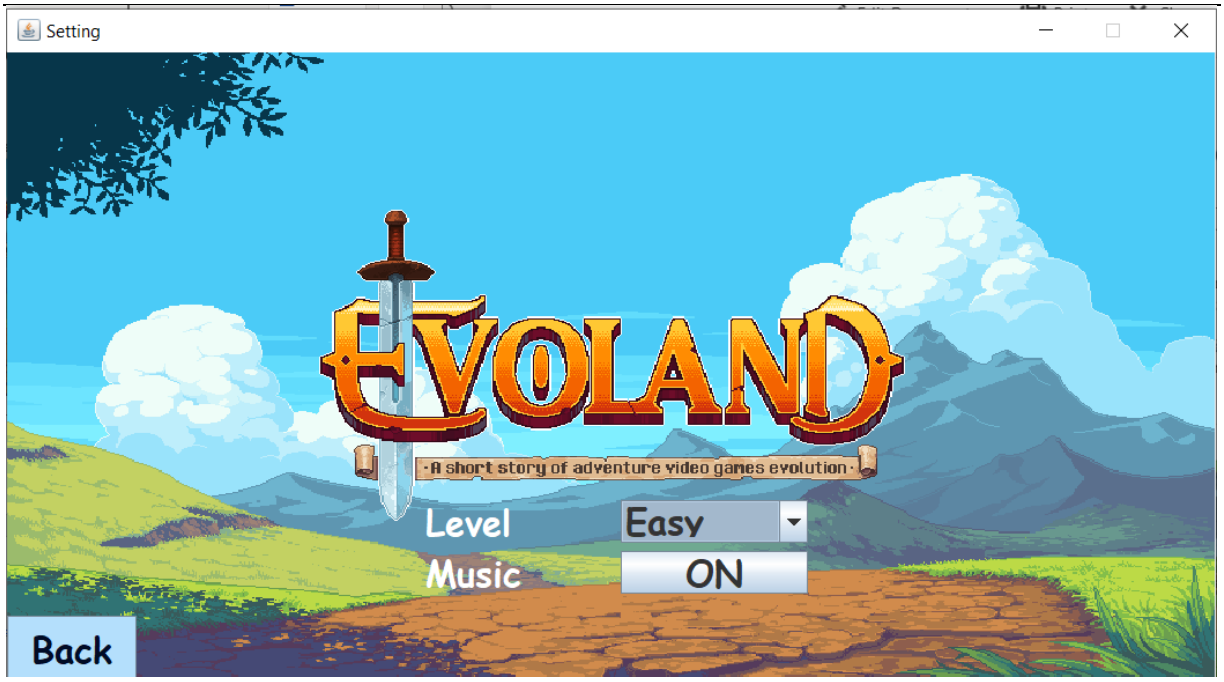
- Thể hiện được bản đồ và các đối tượng (cây cối,nước,đường,….) trên bản đồ trên giao diện đồ họa.
- Người chơi có thể điều khiển nhân vật di chuyển ,tấn công quái vật bằng các phím chức năng.
- Quái vật có khả năng di chuyển tấn công lại nhân vật. Quái vật thì được chia làm hai nhóm chính là quái vật nhỏ (di chuyển dưới đất) và boss (rồng).
- Game gồm 4 bản đồ ,người chơi khi vào game thì sẽ bắt đầu ở bản đồ đầu tiên (bản đồ chính) và có thể đi qua lại giữa các bản đồ bằng các cổng dịch chuyển. 3 bản đồ còn lại sẽ gồm 1 boss và 1 số quai nhỏ. Khi đánh chết quái nhỏ sẽ nhận được các item giúp tăng máu, tăng tấn công và tăng giáp của nhân vật. Người chơi thắng khi đánh bại toàn bộ boss ở các bản đồ,và thua khi player hết máu.
- Game có âm thanh nền (có thể điều chỉnh bật/tắt).
- Game có 2 chế độ chơi là dễ và khó, ở mỗi chế độ thì máu, tấn công ,giáp của quái vật sẽ được thay đổi tương ứng.
- Chức năng bắt đầu lại game khi nhân vật chiến thắng(Người chơi đánh bại 3 quái vật to ở 3 map) hoặc thua(khi nhân vật hết máu).

4.2. Giao diện chương trình

- Màn hình bắt đầu khi vào game:



  - Start Game: nhấn vào để bắt đầu game.
  - Setting: Thay đổi cài đặt âm thanh và chế độ chơi.



- "?": Hiển thị hướng dẫn chơi game.

- Khi Vào Game:
    - Bản đồ đầu tiên:

Bản đồ đầu tiên được mô hình là nơi xuất phát của nhân vật

- Trên bản đồ này sẽ có 2 cổng dịch chuyển đến hai bản đồ còn lại. Khi nhân vật đến gần vị trí dịch chuyển thì người chơi sẽ được dịch chuyển đến bản đồ tương ứng với cổng dịch chuyển đó.

- Bản đồ thứ 2:

- Bản đồ này sẽ có quai nhỏ và boss.
- Khi người chơi đánh chết quai nhỏ thì sẽ nhận được item buff máu ,buff giáp và buff damage như hình và để nhặt và sử dụng các item, người chơi cần điều khiển nhân vật đến gần item và nhấn E.
- Người chơi có thể quay lại bản đồ ban đầu khi đến vị trí dịch chuyển.
- Người chơi điều khiển nhân vật bằng các phím chức năng.

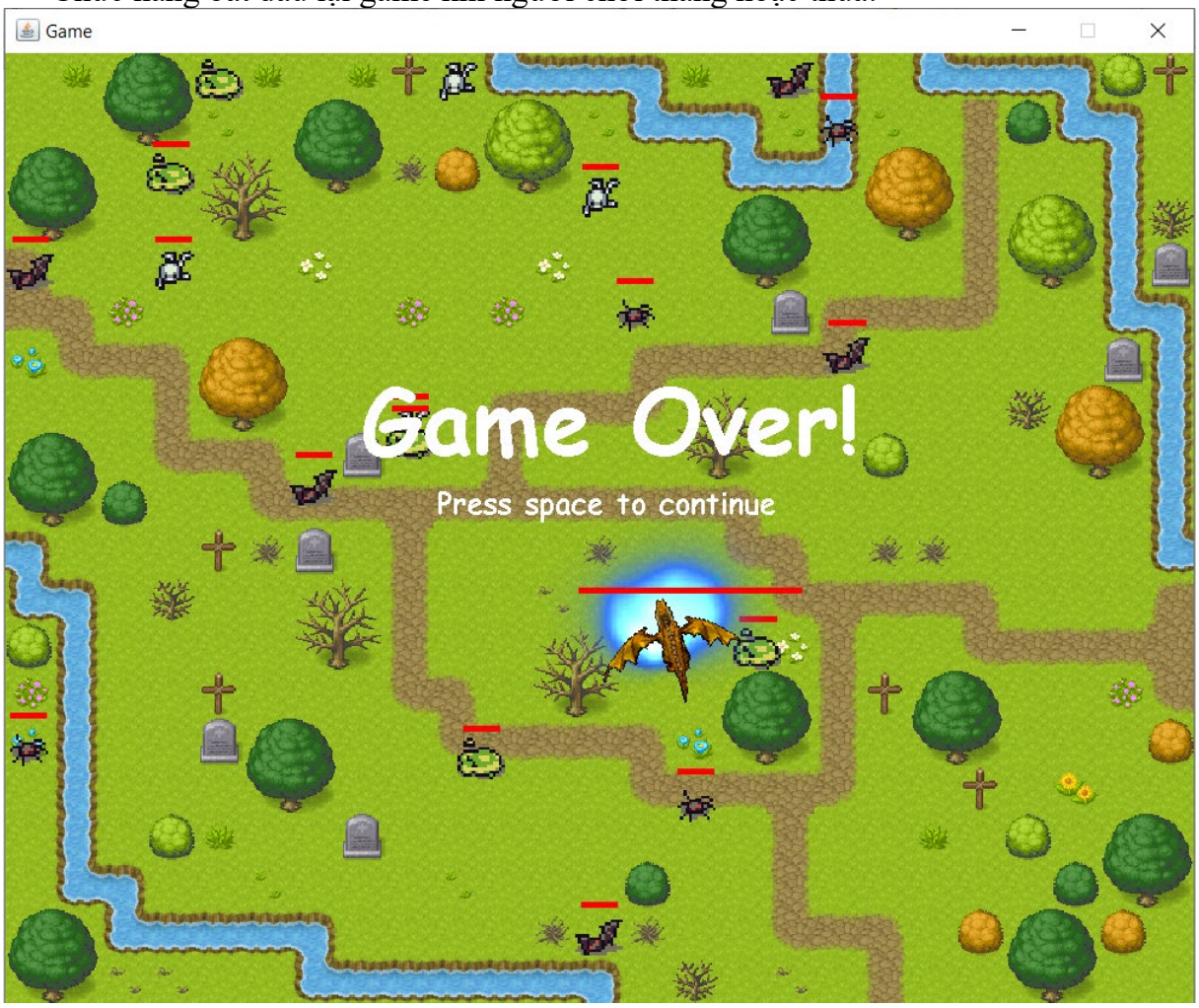
- Bản đồ thứ 3:

  - Tương tự với bản đồ thứ 2.

-

- Bản đồ thứ 4:

- Chức năng bắt đầu lại game khi người chơi thắng hoặc thua:



- Người chơi nhấn phim space để quay lại màn hình bắt đầu game.

4.3. Kiểm thử các chức năng đã thực hiện

4.3.1. Kiểm thử cho chức năng 1

- **Chức năng:** Vẽ bản đồ với các đối tượng (cây, cỏ, đường ,…).

Bảng 1: Kết quả kiểm thử chức năng vẽ bản đồ

STT	input	output	Exception	Kết quả
1	Đọc file map.txt chứa ma trận vị trí đặt các đối tượng trong map	Hiển thị map lên màn hình	Xử Lý Chuẩn	OK
2	Đọc ảnh đối tượng từ file	Ảnh của đối tượng trên map	Không xử lý	OK

4.3.2. Kiểm thử cho chức năng 2

- Chức năng: bật/tắt âm thanh,lựa chọn chế dộ chơi.

Bảng 2: Kết quả kiểm thử chức năng cài đặt âm thanh, chế độ chơi

STT	input	output	Exception	Kết quả
1	Người dùng không lựa chọn gì	Bật / Tắt âm thanh, chế độ chơi Khó/Dễ	Xử Lý Chuẩn	OK
2	Lựa chọn của người dùng	Chế độ tương ứng	Không xử lý	OK

4.3.3. Kiểm thử cho chức năng 3

- Chức năng : Start Game.

Bảng 3: Kết quả kiểm thử chức năng Start Game.

STT	input	output	Exception	Kết quả

1	Lựa chọn của người dùng	Bắt đầu game	Không Xử Lý	OK

4.3.4. Kiểm thử cho chức năng 4

- Chức năng : điều khiển nhân vật tấn công,nhặt đồ :

Bảng 3: Kết quả kiểm thử chức năng tương tác với nhân vật.

STT	input	output	Exception	Kết quả
1	Thao tác của người dùng (khi đang chơi game mà người dùng thao tác lên ứng dụng khác)	Hoạt động của nhân vật trong game	Không xử lý	FAIL
2	Thao tác điều khiển tấn công quái vật	Máu của quái vật bị giảm		OK
3	Thao tác nhặt đồ, và sử dụng đồ	Các chỉ số của nhân vật tăng lên		Ok

4.3.5. Kiểm thử cho chức năng 5

- Chức năng : quái vật di chuyển tấn công nhân vật

Bảng 3: Kết quả kiểm thử chức năng hoạt động của quai vật.

STT	input	output	Exception	Kết quả

1	Khi nhân vật đến gần phạm vi tấn công của boss	Máu của nhân vật bị giảm	Không xử lý	OK
2	Khi nhân vật không nằm trong phạm vi tấn công của boss	Nhân vật không bị tấn công	Không xử lý	OK

4.3.6. Kiểm thử cho chức năng 6

- Chức năng : dịch chuyển giữa các map khi giết được boss

Bảng 6: Kết quả kiểm thử chức năng dịch chuyển giữa các map.

STT	input	output	Exception	Kết quả
1	Boss bị tiêu diệt	Cổng dịch chuyển giữa các map được mở, người chơi có thể sử dụng cổng dịch chuyển		OK
2	Boss chưa bị tiêu diệt	Cổng dịch chuyển chưa được mở		OK

4.3.7. Kiểm thử cho chức năng 7

- Chức năng : nhấn phim space để quay lại màn hình start game khi chiến thắng hoặc thua

Bảng 7: Kết quả kiểm thử chức năng quay lại màn hình bắt đầu game.

STT	input	output	Exception	Kết quả
1	Bắt sự kiện nhấn phim space từ người chơi	Hiện màn hình bắt đầu game		OK

4.3.8. Kết luận

- Chương trình chạy ổn định ,tuy nhiên còn một số chỗ code chưa được tối ưu. Chưa quản lý các luồng của trò chơi một cách tối ưu (có thể còn tồn tại một số luồng không cần thiết chạy ngầm khi game đang hoạt động)

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Qua một hành trình dài 4 tháng, bài tập lớn lần này đã đem đến một đề tài bổ ích và thú vị cho các thành viên nhóm 16. Các thành viên đã cùng nhau tìm hiểu về lập trình hướng đối tượng, kĩ năng lập trình game và đặc biệt các thành viên được thầy truyền đạt kiến thức một cách chi tiết, tận tâm. Qua đó từng thành viên đã có thể biết được quá trình hoàn thành 1 dự án game, biết cách phân tích, đặc tả và thiết kế bài toán, và biết được các kiến thức quan trọng trong môn học để hoàn thiện sản phẩm một cách tốt nhất.

Sản phẩm game lần này đã đáp ứng được những tiêu chí cơ bản về một game sinh tồn như xây dựng giao diện mở đầu, hiệu ứng kết thúc, xây dựng cốt truyện, quá trình chơi của nhân vật chính phải cố gắng tồn tại và đạt được mục tiêu để đi đến chiến thắng. Đặc biệt nhóm đã phát triển thêm nhiều tính năng như xây dựng 3 màn chơi, xây dựng từng map đẹp và tỉ mỉ, hệ thống quái thông minh riêng cho từng màn, hệ thống items đa dạng, và các hiệu ứng di chuyển, va chạm, tấn công cho nhân vật, quái và nhiều hiệu ứng khác như dịch chuyển, kết thúc với mong muốn trải nghiệm game được tốt nhất.

Nhược điểm đặt ra là còn một số chỗ code chưa được tối ưu. Chưa quản lý các luồng của trò chơi một cách tối ưu. Bên cạnh đó đồ họa của trò chơi chưa được xử lý một cách đẹp nhất

Hướng phát triển được đề ra là có 2 mục tiêu. 1 là sửa lỗi đã gặp trong quá trình chạy chương trình, quản lý các luồng của trò chơi một cách tốt nhất. 2 là phát triển thêm nhiều hiệu ứng, thuộc tính đặc biệt cho game, cải thiện hình ảnh nhân vật có thêm độ sắc nét cao, hình thành 1 sản phẩm hay, độc, lạ và ổn định.

Trong quá trình làm game không tránh khỏi những sai sót và hạn chế. Vì thế nhóm mong có thể có được những nhận xét, góp ý chỉnh sửa của thầy về bài tập lớn lần này. Đồng thời nhóm gửi lời cảm ơn đến thầy đã dạy dỗ tận tâm, giúp đỡ và mang đến trải nghiệm tuyệt vời này cho nhóm chúng em.

< Thanks for reading. Have a great day! >

TÀI LIỆU THAM KHẢO

[1] Java AWT Tutorial – www,javatpoint.com – (2011-2018)
[2] Giáo án môn Lập trình hướng đối tượng – thầy Nguyễn Mạnh Tuấn – 2021
[3] JavaGame tutorial - code-knowledge.com
[4] Class diagram & Use case diagram - staruml.io

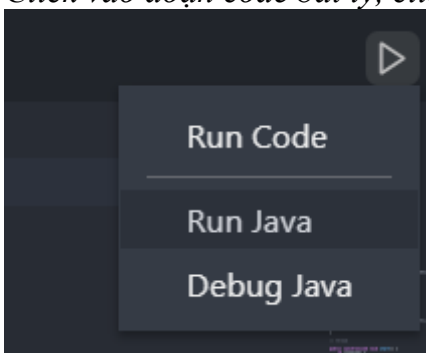
PHỤ LỤC

<Phần này đưa ra hướng dẫn cài đặt, hướng dẫn sử dụng của chương trình, một số các vấn đề khác muốn trình bày…>

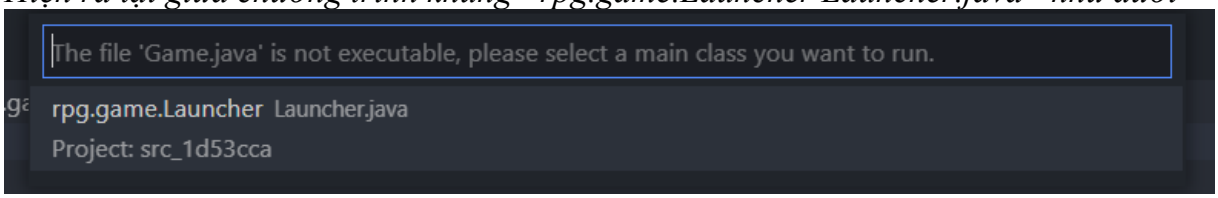
+ Nên tải bản word và powerpoint về máy để có thể đọc rõ và đẹp nhất
+ Mở file game: "Newgame" tại các IDE có hỗ trợ và đã cài đặt Java và JDK.
+ Chạy chương trình tại Launcher

+ Hướng dẫn cách chạy game tại VSCode để có độ ổn định tuyệt đối

Click vào đoạn code bất lỳ, click vào "run java" ở bên phải màn hình



Hiện ra tại giữa chương trình khung "rpg.game.Launcher Launcher.java" như dưới



Click vào khung đó, đọc hướng dẫn và bắt đầu trải nghiệm game