TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI

UNIVERSITY OF TRANSPORST AND COMMUNICATIONS

BÁO CÁO BÀI TẬP LỚN BỘ

MÔN CÔNG NGHỆ JAVA

ĐỀ TÀI: LẬP TRÌNH GAME 2D

Giảng viên giảng dạy : **(thầy) Vũ Huấn**

Lớp : **Công nghệ thông tin 6K62**

Sinh viên thực hiện : **Nguyễn Tiến Tùng**

Mã sinh viên : **211200893**

Hà Nội 2023

LỜI MỞ ĐẦU

Hiện nay nhu cầu giải trí ngày càng phát triển, ngành công nghệ thông tin ngày càng đa dạng các lĩnh vực hơn. Trong đó ngành phát triển game là một trong những ngành càng ngày cảng phát triển, ngày càng nhiều game mới được ra mắt đáp ứng nhu cầu giải trí của con người. Để hiểu về nguyên lý, các bước phát triển một game em đã quyết định tự mình học và code ra một game 2D bằng ngôn ngữ lập trình java.

Chắc chắn rằng, với kinh nghiệm cũng như kiến thức còn non trẻ, nên game của mình vẫn còn nhiều thiếu sót và chưa hoàn thiện. Vì vậy em rất mong nhận được nhận xét từ thầy. Để game được thêm hoàn thiện.

Em xin trân thành cảm ơn thầy trong suốt quá trình giảng dạy đã truyền đạt cho em kiến thức để em có thể có thể thực hiện dự án của mình.

Mục lục

1. Giới thiệu tổng quan của trò chơi

Hiện tại trò chơi vẫn chưa có tên, nên em sẽ đặt tên cho game là Game Big hat knight. Sở dĩ game có tên như vậy xuất phát từ đặc điểm ngoại hình nhân vật, nhân vật xuất hiện với chiếc mũ to, mang trên mình thanh kiếm nên em đặt tên game là Big hat knight.

* Game đáp ứng các yêu cầu:

- Thiết kế theo hướng đối tượng

- Làm việc với tệp tin (đọc file ảnh, đọc file âm thanh)

- Thao tác với cơ sở dữ liệu, dùng My SQL để lưu vị trí nhân vật, máu nhân vật, map

- Đa luồng: có luồng âm thanh, luồng logic tạo vòng lặp game.

- Có bắt lỗi

- Có GUI

* Các tính năng của trò chơi:

- Game bao gồm giao diện đồ hoạ thân thiện người dùng.

- Game có các đối tượng như: nhân vật, kẻ thù, map, các item tăng máu, các menu trạng thái game.

- Game có kết thúc game: ở mỗi màn chơi thì sẽ có một map bản đồ riêng, có số lượng kẻ thù khác nhau, sau khi nhân vật tiêu diệt xong kẻ thù thì sẽ kết thúc màn và sẽ được chuyển sang màn mới. Game được thiết kế có 3 màn. Sau khi kết thúc màn cuối thì game sẽ kết thúc.

- Game có âm thanh, các màn khác nhau có âm thanh khác nhau, từng trạng thái của nhân vật đều có một loại âm thanh khác nhau.

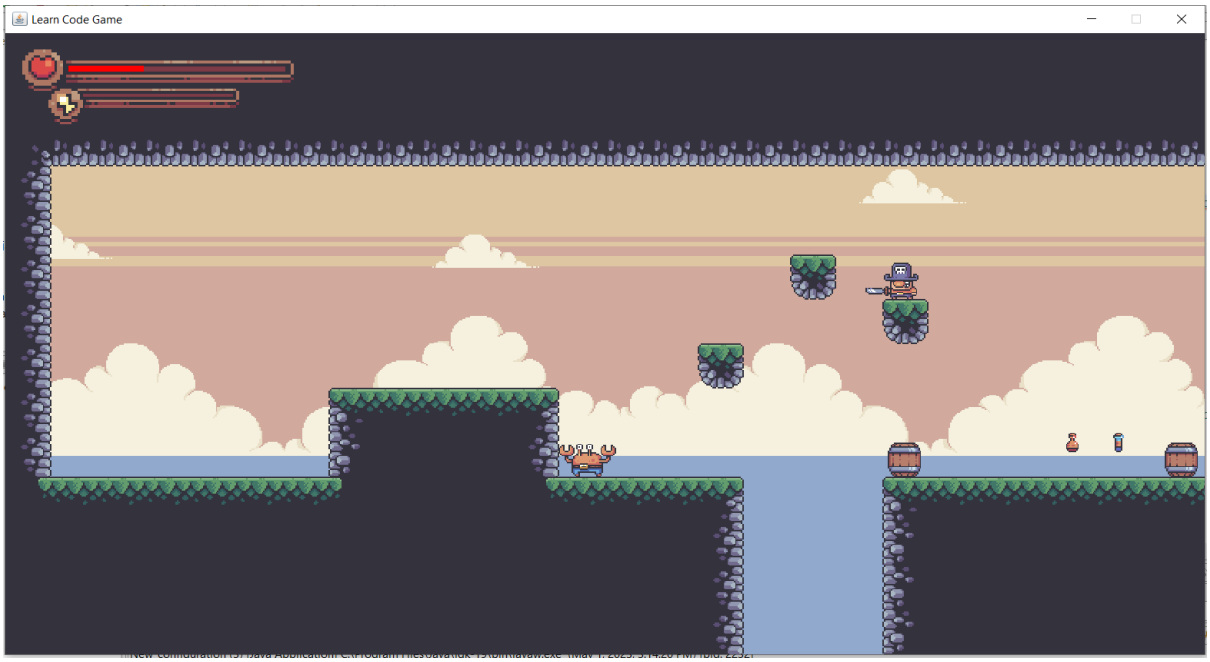
- Game có chức năng lưu game, khi chúng ta kich vào nút lưu game thì dữ liệu về toạ độ nhân vật, máu, vị trí nhân vật trong map lên SQL. Lần sau khi game chạy lên ta có thể tiếp tục chơi.

2. Giao diện game, ý tưởng code giao diện game

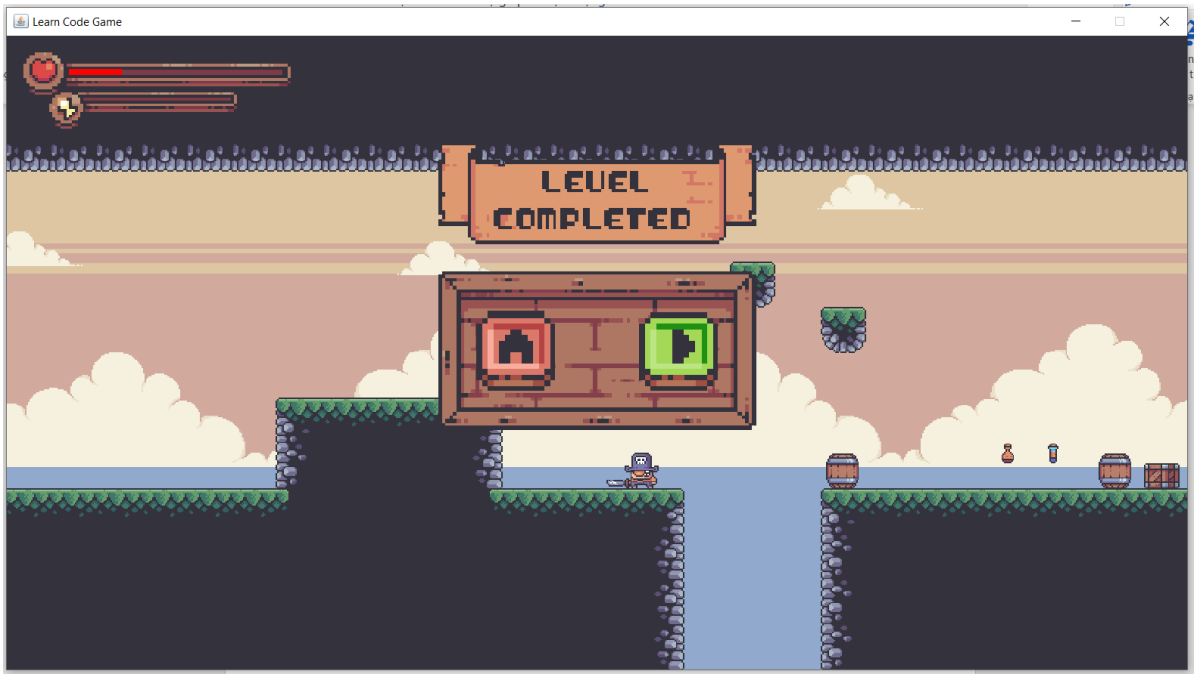
2.1. Giới thiệu giao diện game

Giao diện trò chơi gồm có các màn, nhân vật, kẻ thù, và các menu trạng thái của game.

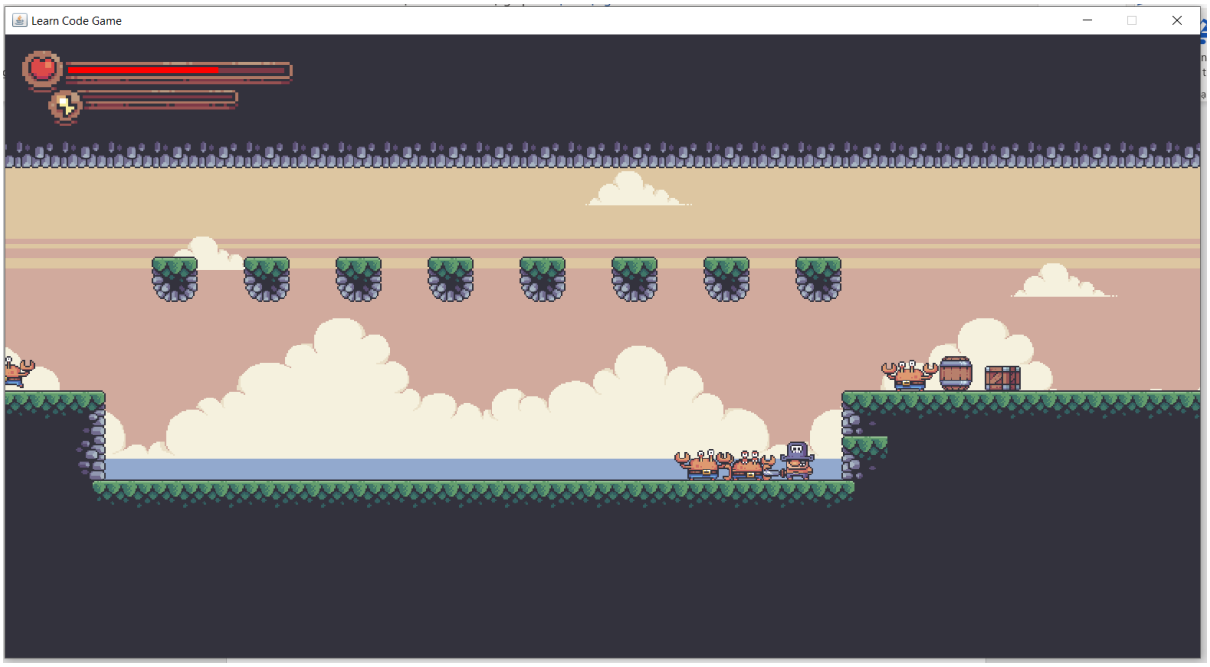
Giao diện Menu chính



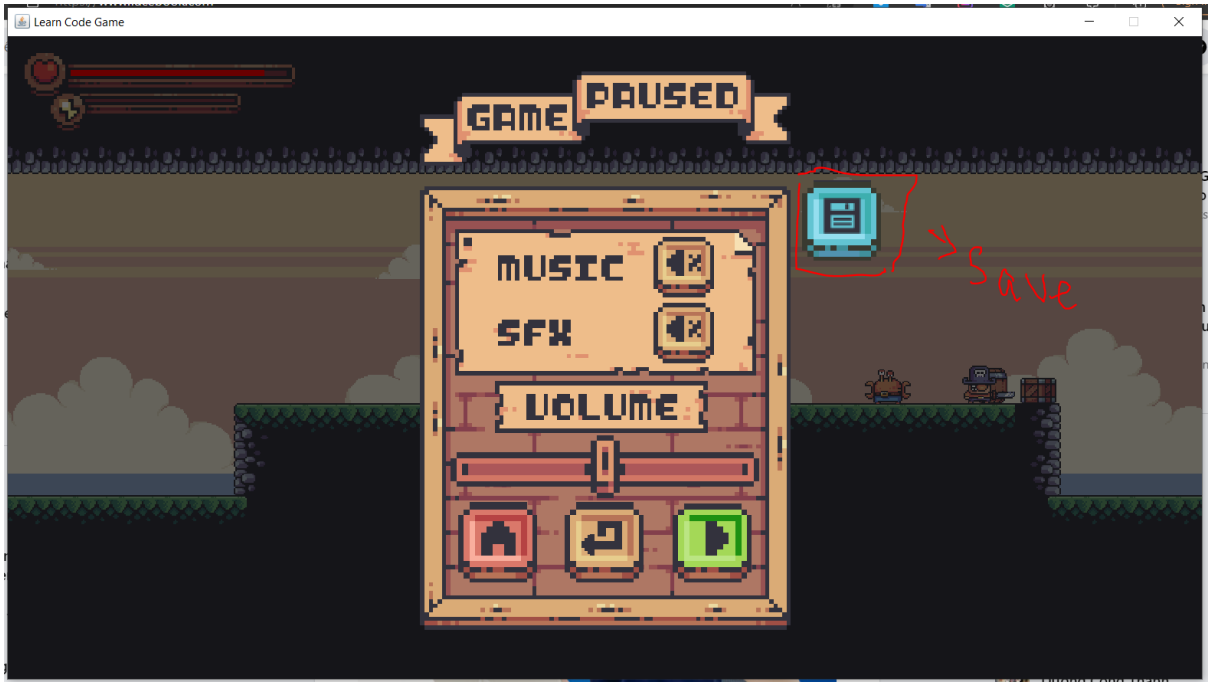
Giao diện map 1



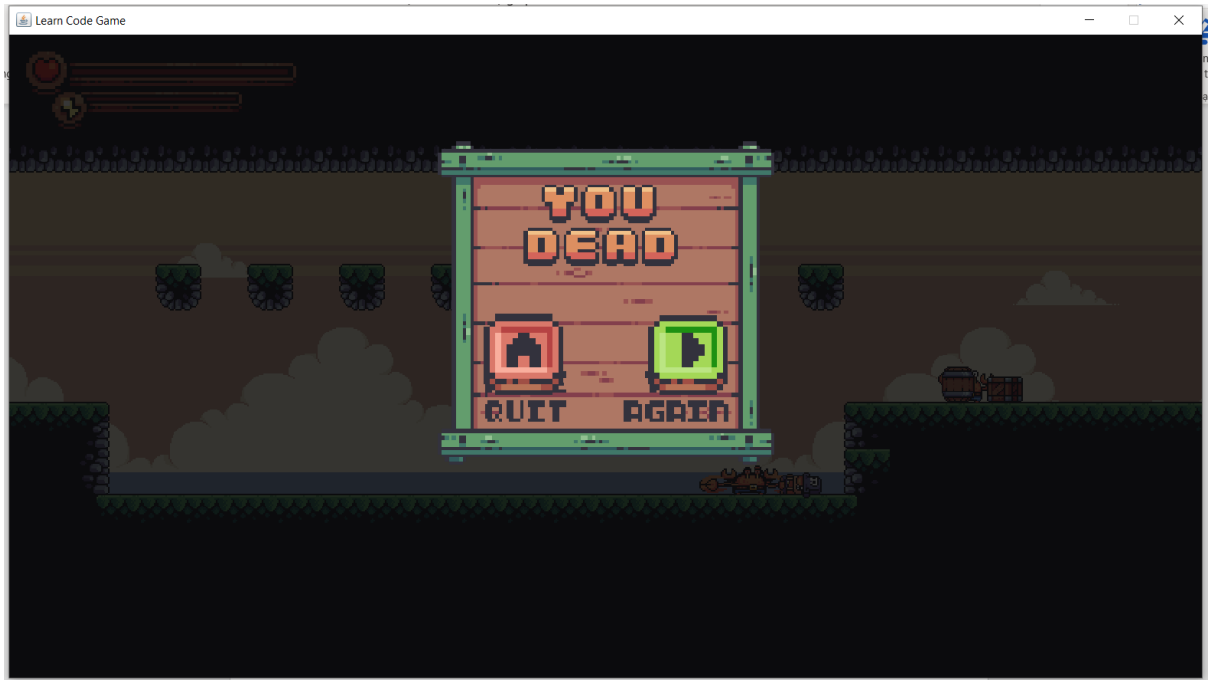
Menu hoàn thành map 1



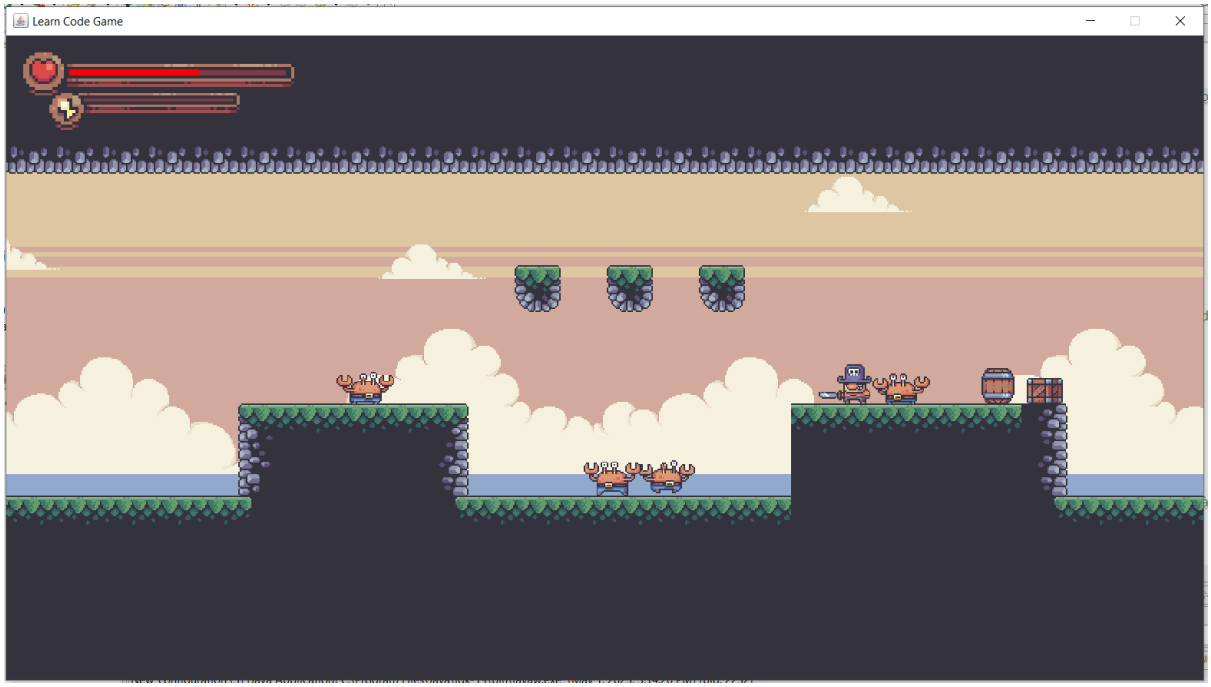
Giao diện map 2



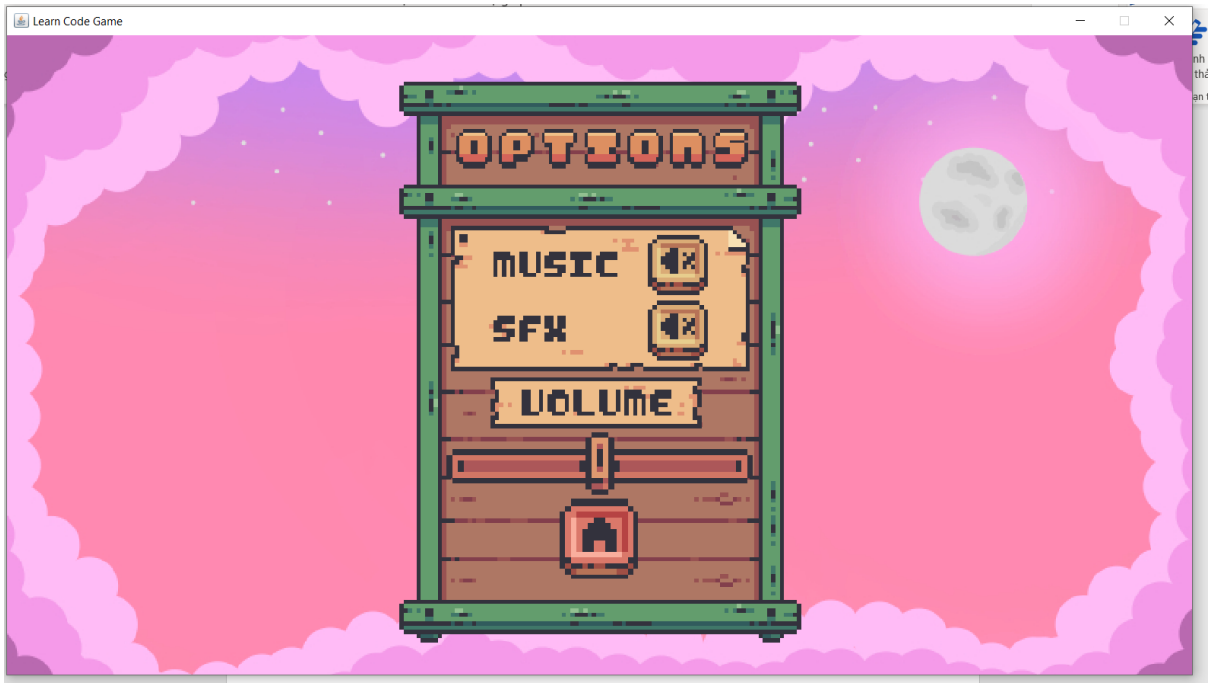
Giao diện Pause Game



Menu khi nhân vật chết



Giao diện map 3



Giao diện game Option

2.2. Cách thức hoạt động của giao diện

Giao diện Menu chính

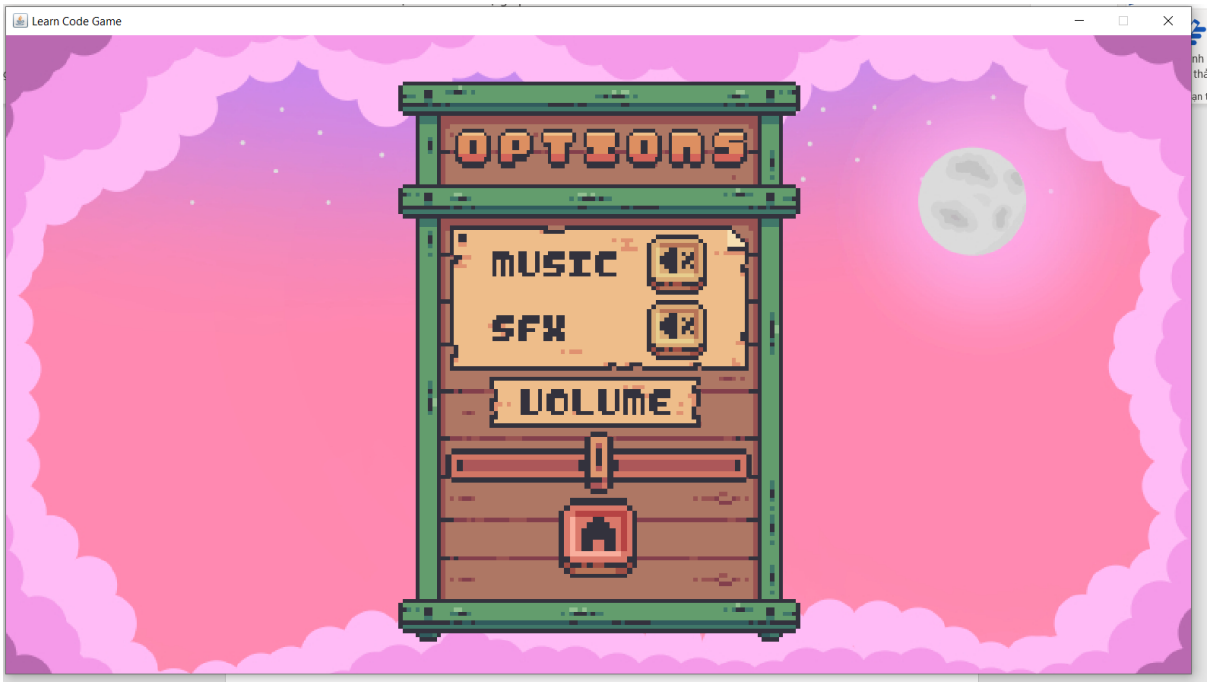
a) Giao diện Menu chính

Khi bắt đầu vào game thì Menu chính của game hiện ra, tại đây có các lựa chọn Continue, Newgame, Option, Quit.

Nếu chúng ta nhấn continue thì chúng ta sẽ tiếp tục chơi game tại vị trí mà ta đã lưu từ trước.

Nếu chúng ta nhấn NewGame thì toàn bộ game của chúng ta sẽ reset lại từ đầu. Từ máu nhân vật, vị trí nhân vật, map.

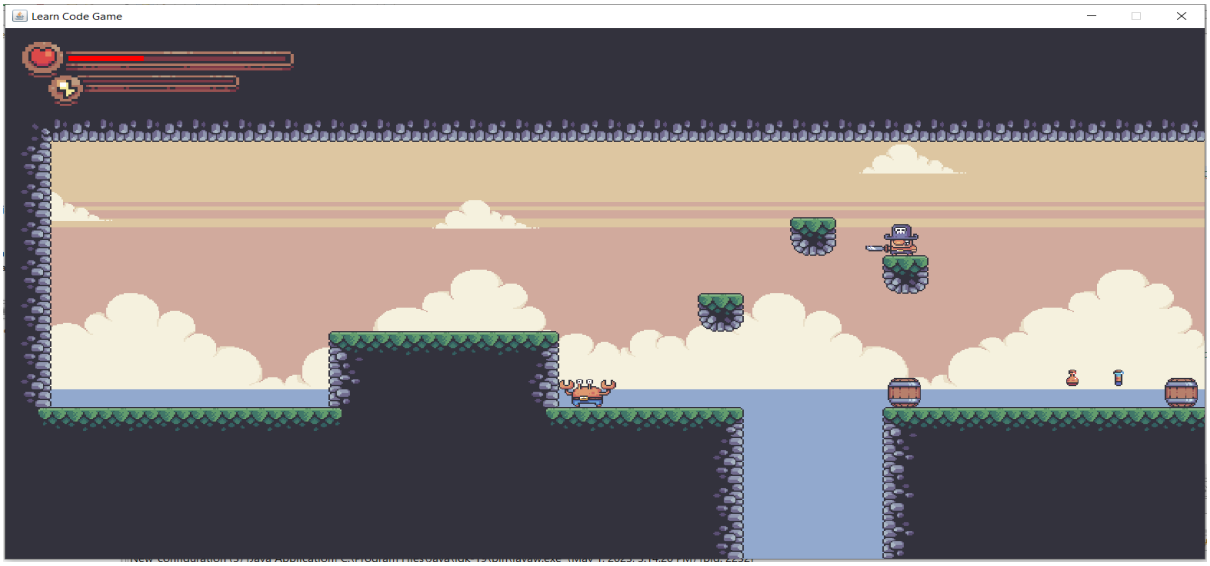
Nếu chúng ta nhấn vào Option thì bảng cài đặt sẽ hiện ra tại đây chúng ta có thể tuỳ chỉnh âm thanh có thể bật tắt, điều chỉnh âm lượng, bặt tắt âm thanh hiệu ứng nhân vật bằng các tương tác với button SFX, bật tắt âm thanh nền bằng cách tương tác với button music. Thay đổi âm lượng bằng cách kéo thả volume.



Giao diện game Option

Nếu chúng ta nhấn vào Quit thì sẽ thoát game.

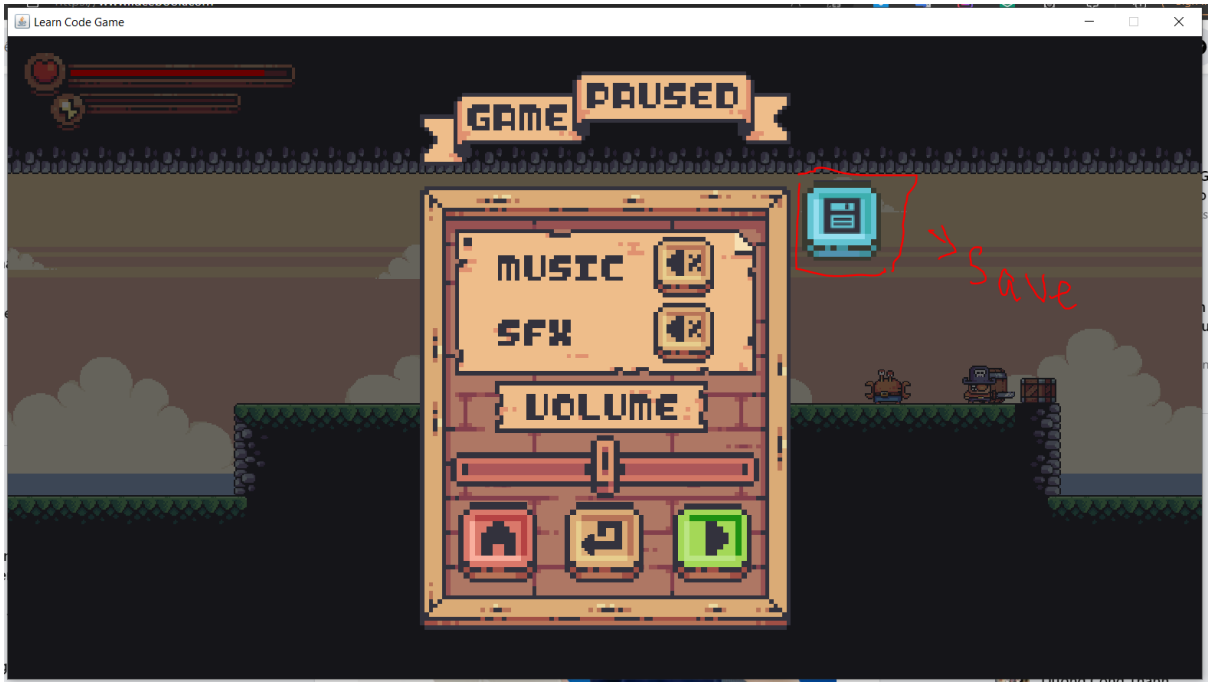
b) Giao diện các màn, pause game và hoàn thành màn chơi, game over



Giao diện map 1

Mỗi một màn thì sẽ có map bản đồ cho các màn, trong màn sẽ có kẻ thù vào các item tăng máu cho nhân vật.

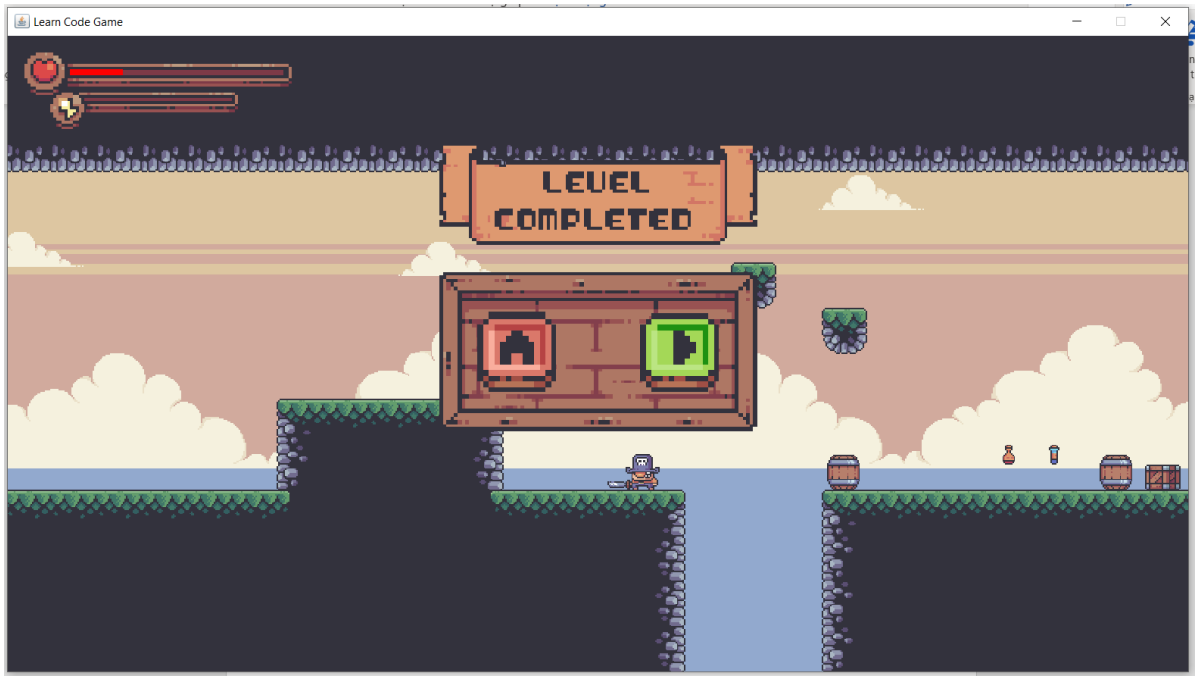
Trong mỗi màn nếu chúng t muốn tạm ngừng thì có thể nhấn phím Escapse và giao diện PauseGame hiển thị



Giao diện Pause Game

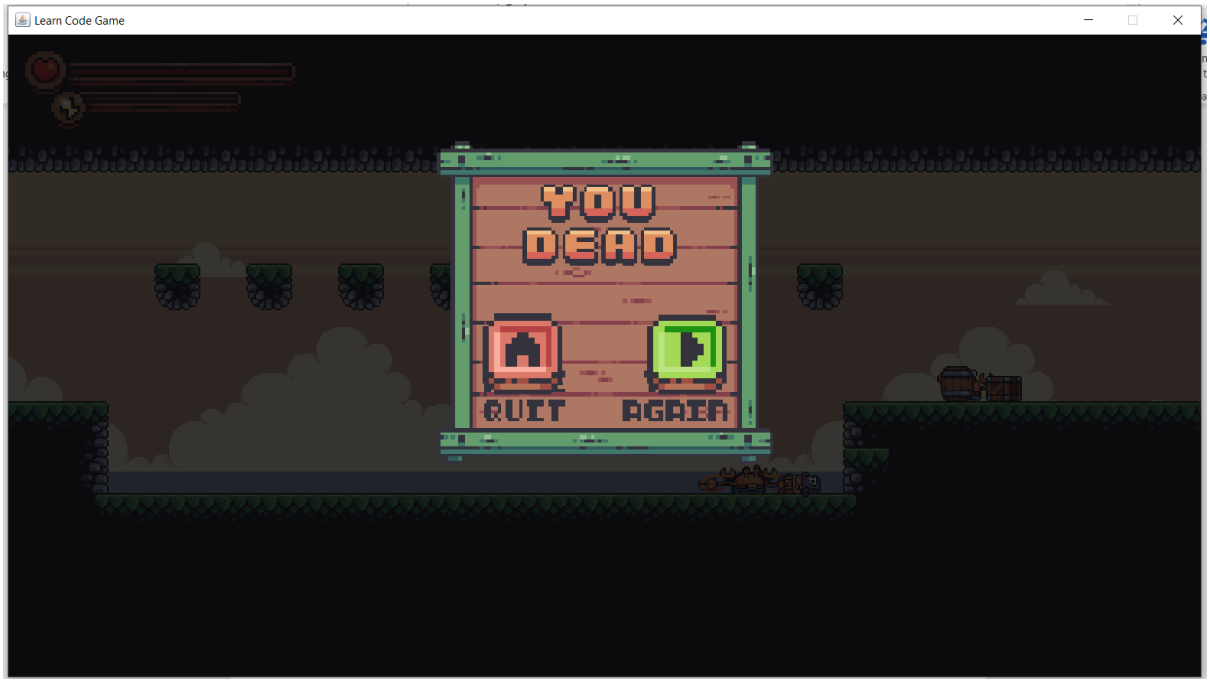
Tại đây người chơi có thể thay đổi trạng thái âm thanh, có nút tiếo tục chơi, có nút chơi lại từ đầu của màn này, có nút quay trở lại Menu chính, có nút save game.

Sau khi nhân vật chém chết hết kẻ thù trong màn thì hiển thị một bảng Menu với hai lựa chọn đi tới màn tiếp hoặc quay trở về Menu chính.



Menu hoàn thành map 1

Nếu như nhân vật để kẻ thù giết chết thì sẽ hiển thị giao diện game over



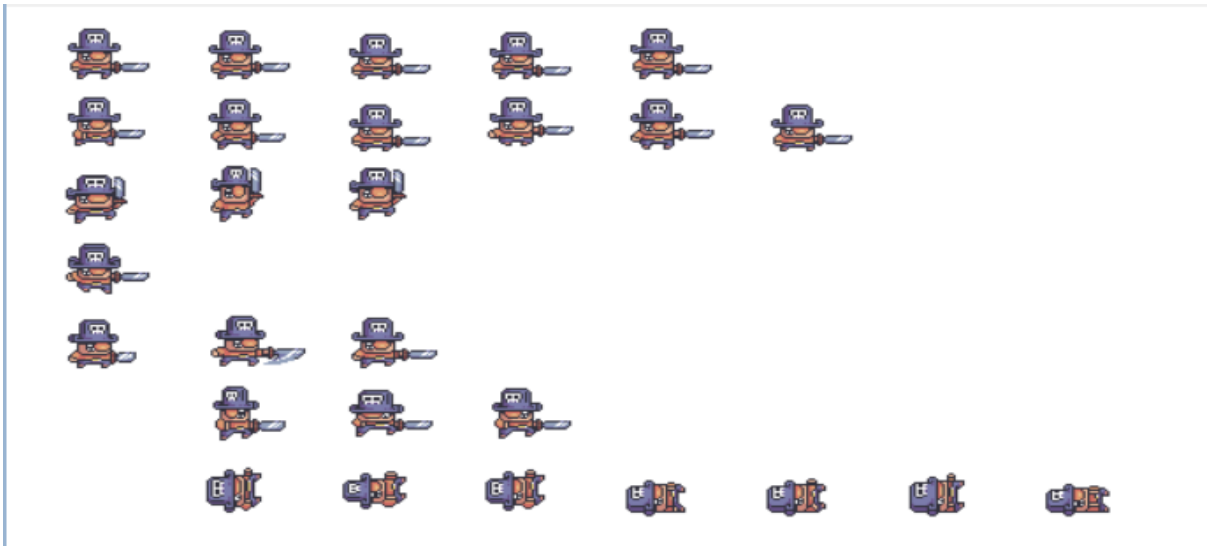
Menu khi nhân vật chết

2.3. Giải thích cách vẽ giao diện

2.3.1. Giải thích cách vẽ nhân vật và kẻ thù

B1: Ta lưu các hoạt ảnh trạng thái của nhân vật, kẻ thù vào mảng ảnh 2 chiều. Trong đó mảng hai chiều này một chiều có chỉ số là trạng thái của nhân vật một chiều là dòng trạng thái của nhân vật

```
public void loadStatus() {//tải trạng thái của nhân vật
        BufferedImage imgMainer = Map.GetMap(Map.PLAYER_STATUS);// lấy một ma trận trong đó mỗi mảng 1 chiều chứa các hoạt ảnh của các trạng thá
        statusOfMainer = new BufferedImage[7][8];// khai báo một ma trận để chứa tất cả các hoạt ảnh của nhân vật
        for(int j=0;j<statusOfMainer.length;j++)
            for(int i=0;i<statusOfMainer[j].length;i++){
            statusOfMainer[j][i] = imgMainer.getSubimage(i*64,j* 40,64,40);// lưu hoạt ảnh vào ma trận
            }
```



B2: Ta cập nhật các trạng thái của nhân vật tuỳ thuộc vào các tương tác sự kiện bàn phím, chuột của người dùng

```
    public void render(Graphics g, int levelOffset) {// reset lai hộp hitbox
        g.drawImage(statusOfMainer[status][statusIndex] ,
                (int) (hitBox.x-xDrawOffset)-levelOffset+flipX,// lí do cần cộng thêm
                (int)( hitBox.y-yDrawOffset),
                width*flipW,height, null);//getFocus is image of server
        // tai sao cần nhân thêm flipW
                drawStatusPower(g);
//              drawActtackBox(g,levelOffset);
//              drawHitBox(g, levelOffset);
    }
```

Tại một tương tác sự kiện thì sẽ truyền vào các key Input tại mỗi key input ta sẽ thiết lập các biến logic trạng thái tương ứng cho nhân vật.

Sau đó chỉ số StatusIndex sẽ được chạy và hoạt ảnh của nhân vật được vẽ ra. Như vậy hoạt ảnh của nhân vật sẽ được vẽ liên tục cho từng trạng thái. Nhân vật luôn ở trạng thái động.

```
private void updateStatusOfMainer() {
    statusTick++;// biến này là biến thời gian để cha
    if(statusTick>=STATUSSPEED) {
        statusTick=0;
        statusIndex++;
        if(statusIndex>=GetSpriteAmount(status)) {//
            statusIndex=0;
            isActtack=false;// khi chạy hết hoạt ảnh
            acttackChecked=false;// khi đã tấn công x
        }
    }
```

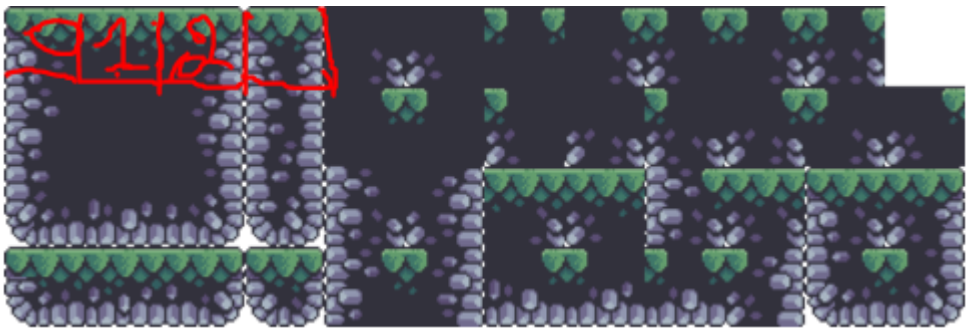
Chạy các hoạt ảnh trong 1 dòng trạng thái của nhân vật

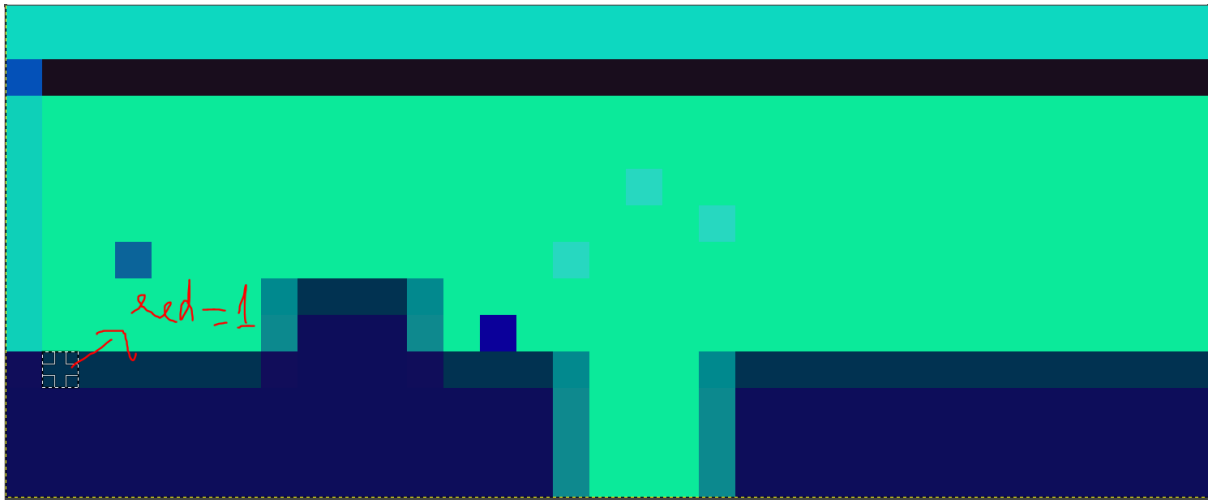
Ví dụ: khi ta nhấn nhảy thì biến trạng thái mà chúng ta thiết lập cho trạng thái của nhân vật là nhảy sẽ chuyển thành true, sau đó trạng thái của nhân vật sẽ sẽ truyền thành trạng thái nhảy và status sẽ được chuyển sang trạng thái nhảy tức giá trị của status = 2

```java
public static class ControllerPlayer{// hành động của nhân vật đây cho dòng hoạt ảnh của nhân vật là dòng bao nhiêu
    public static final int RUNNING=1;
    public static final int IDLE=0;
    public static final int JUMP=2;
    public static final int FALLING=3;
    public static final int HIT=5;
    public static final int DEAD=6;
    public static final int ACTTACK=4;
    public static int GetSpriteAmount(int playerAction) {//số lượng hoạt ảnh của các trạng thái
        switch(playerAction) {// đây là số hoạt ảnh của 1 mảng trạng thái của nhân vật
        case DEAD:
            return 8;
        case RUNNING:
            return 6;
        case IDLE:
            return 5;
        case HIT:
            return 4;
        case JUMP:
        case ACTTACK:
            return 3;
        case FALLING:
```

2.3.2. Giải thích các vẽ map

Em dùng phương pháp vẽ map dựa vào mảng chỉ số màu của các ô. Màu xanh em định danh cho kẻ thù, màu đỏ em vẽ map. Nguyên lý của phương pháp này đó là em định danh cho tất cả các phần từ của map là một số nguyên. Sau đó em sẽ vẽ một lưới ma trận các ô pixel và tuỳ thuộc vào vị trí nào trong ma trận chúng ta sẽ đặt các phần tử map như thế nào mà em sẽ thiết lập màu cho các ô pixel như vậy. Ví dụ các phần tử trên nền của game em sẽ để giá trị màu đỏ của nó là 1. Thì em sẽ vẽ các ô trên nền có giá trị màu đỏ là 1.





```
public static int [][] GetLevelData(BufferedImage img){// tạo 1 mảng xong gán id của bức một hoạt ảnh nhỏ cho mảng
    int[][] lvlData = new int[img.getHeight()][img.getWidth()];
    for (int j = 0; j < img.getHeight(); j++)
        for (int i = 0; i < img.getWidth(); i++) {
            Color color = new Color(img.getRGB(i, j));
            int value = color.getRed();
            if (value >= 48)
                value = 0;
            lvlData[j][i] = value;
        }
    return lvlData;
}
```

```
private void importMap() {
    BufferedImage img = Map.GetMap(Map.Level_Map);// lấy ảnh chứa tất cả các ô để xây dựng map
    levelMap = new BufferedImage[48];// vì ảnh chứa 48 ô để dựng map nên có tổng 48 level cho map tức là có 48 loại ô
    for(int j=0;j<4;j++)
        for(int i=0;i<12;i++) {
            int index =j*12+i;
            levelMap[index] = img.getSubimage(i*32, j*32, 32,32);// cho các loại ô này lưu vào mảng, mảng này chứa các subImage;
        }
}
public void draw(Graphics g,int levelOffset) {// có thể tưởng tượng bản chất khung hình đang đứng yên và chính những hoạt ảnh nhỏ đar
    for(int j=0;j<Game.TILES_IN_HEIGHT;j++)
        for(int i=0;i<levels.get(levelIndex).getLevelData()[0].length;i++) {// ta cho man hình jfram co kich co la ma tran tổng ô nar
            int index = levels.get(levelIndex).getMapIndex(i, j);//lấy trị số id của khung hình;
            g.drawImage(levelMap[index], i*Game.TILES_SIZE - levelOffset,j*Game.TILES_SIZE,Game.TILES_SIZE,Game.TILES_SIZE , null);
        }
```

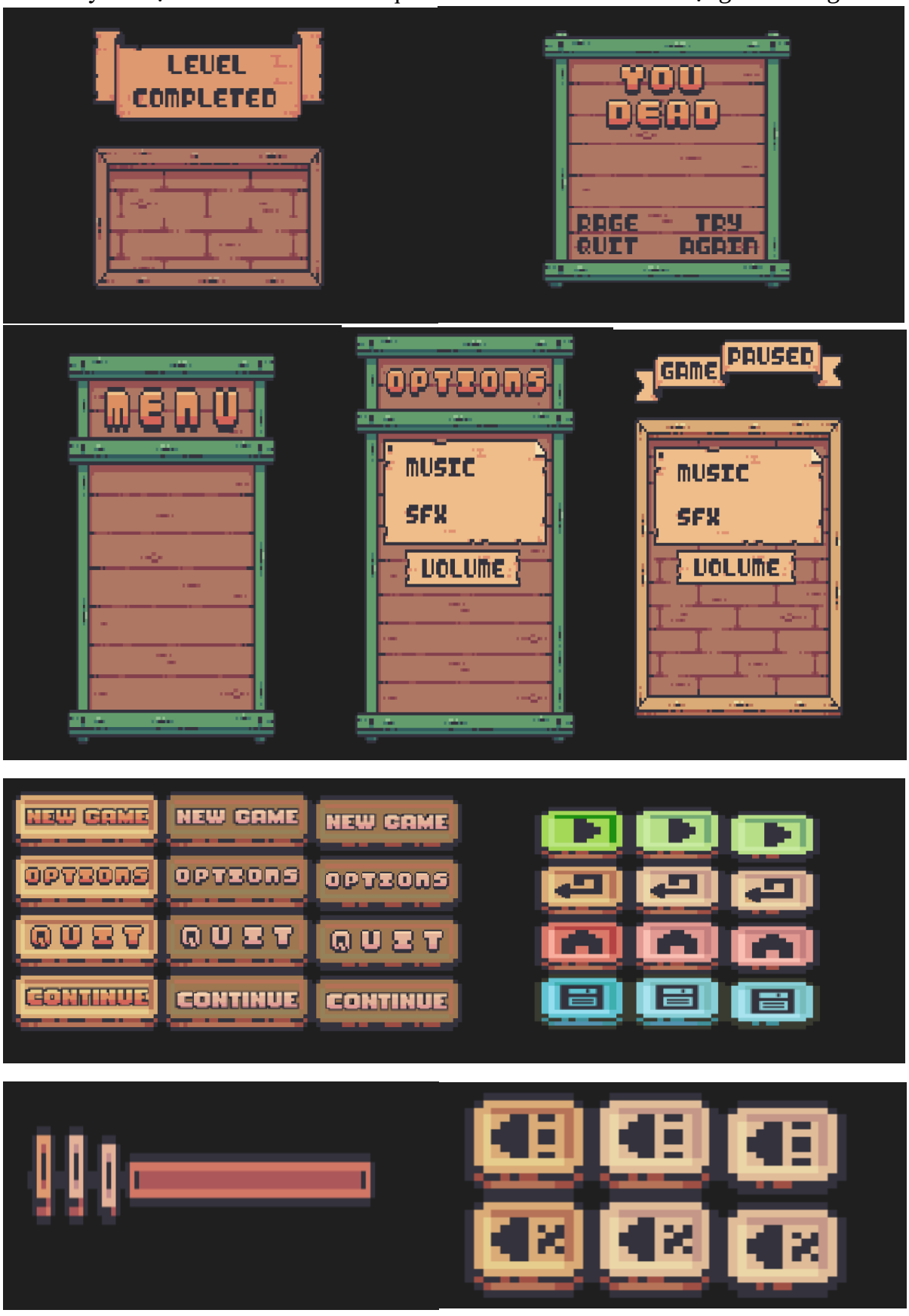
```
public static ArrayList<Crabby> GetCrabbies(BufferedImage img){
    ArrayList<Crabby> list= new ArrayList<>();
    for (int j = 0; j < img.getHeight(); j++)
        for (int i = 0; i < img.getWidth(); i++) {
            Color color = new Color(img.getRGB(i, j));
            int value = color.getGreen();
            if (value == CRABBY)
                list.add(new Crabby(i*Game.TILES_SIZE, j*Game.TILES_SIZE));
        }
    return list;

}
```

Thiết lập kẻ thù cho Map

2.3.3 Giải thích cách vẽ các trạng thái menu như pause game, menu chính, game over

Ta dùng các bức ảnh có sẵn rồi dùng phương thức draw trong thư viện GUI để vẽ các bức ảnh.

Dưới đây là một số bức ảnh cần cho quá trình thiết kế các menu trạng thái của game.



Trước tiên chúng ta cần load các file ảnh, và vẽ nó lên màn hình, tuỳ thuộc vào các trạng thái game mà các menu sẽ hiện khác nhau.

```
private void loadImg() {
    backGrImg=Map.GetMap(Map.MENU_BACKGROUND_GAME);
    optionBackGr=Map.GetMap(Map.OPTION_MENU);
    bgW=(int) (optionBackGr.getWidth()*Game.SCALE);
    bgH=(int) (optionBackGr.getHeight()*Game.SCALE);
    bgX=Game.WIDTH_SIZE/2-bgW/2;
    bgY=(int) (33*Game.SCALE);

}

public void draw(Graphics g) {
    g.drawImage(backGrImg, 0, 0, Game.WIDTH_SIZE, Game.HEIGHT_SIZE, null);
    g.drawImage(optionBackGr, bgX, bgY, bgW, bgH, null);
```

Tiếp theo tuỳ thuộc vào mỗi menu mà chúng ta sẽ phải tạo thêm cho cho các button khác nhau

```
private void createSoundButton() {
    int soundX = (int) (450 * Game.SCALE);
    int soundY = (int) (140 *Game.SCALE);
    int sfxY = (int) (186 * Game.SCALE);
    musicButton = new ButtonSound(soundX, soundY,BTN_SOUND_SIZE ,BTN_SOUND_SIZE );
    sfxButton = new ButtonSound(soundX, sfxY, BTN_SOUND_SIZE, BTN_SOUND_SIZE);
    }
```

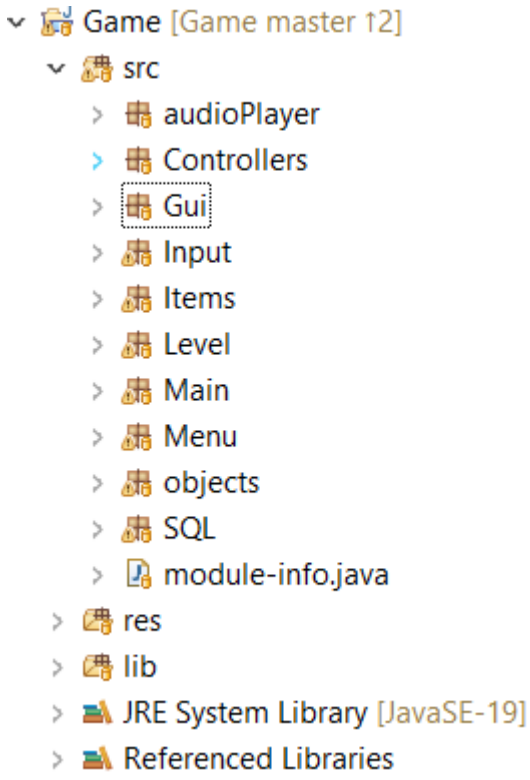
Tiếp theo là vẽ các button lên giao diện menu

```
public void draw(Graphics g) {
    // vẽ menu
    g.drawImage(PauseGameBackground, xBgr, yBgr, BgrWidth, BgrHeight, null);
    // vẽ button
    menuB.draw(g);
    relayB.draw(g);
    unPauseB.draw(g);
    saveB.draw(g);
    audioOptionMenu.draw(g);
}
```

Trên đây là một khuôn mẫu để code các menu trạng thái game. Các menu trạng thại khác trong game được code tương tự.

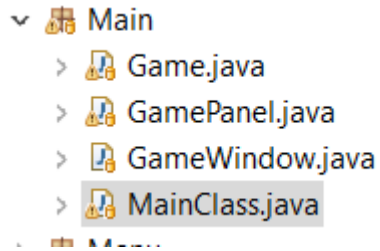
3. Mô hình các gói, các lớp của game

Game được thiết kế chia nhỏ thành nhiều package khác nhau, mỗi package có các lớp với chức năng khác nhau.



* Package chính là Main

Trong package Main có 4 lớp chính: đó là lớp game, lớp game window, lớp gamepanel, lớp main



- Lớp game là lớp để khởi tạo các trạng thái của game, vòng lặp cho game bao gồm vòng lặp về vẽ hình, vòng lặp update, khởi tạo trình điều khiển cho game, khởi tạo cơ sở dữ liệu.

- Trong lớp GamePanel: khởi tạo các trình xử lý sự kiện cho game như KeyListener, MouseListener

- Trong lớp GameWindow: khởi tạo khung hình Jframe cho game.

- Trong lớp Main: nơi khởi tạo game lớp chính của game.

* Package Object: Dùng để tạo các đối tượng cho game, như nhân vật, kẻ thù

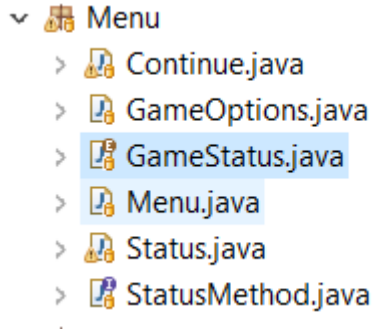
- Lớp Object: là lớp Abstracts có các thuộc tính chung cho nhân vật và kẻ thù như vị trí, các hộp sát thương, các trạng thái…

- Các lớp Player, Enemy: là các lớp được kế thừa từ lớp Object có các thuộc tính riêng của từng loại nhân vật sẽ khác với kẻ thù.

- Lớp Crabby là lớp kế thừa từ lớp Enemy: đây là lớp dùng để khởi tạo các kẻ thù cụ thể cho game vì game sẽ có nhiều loại kẻ thù.Mỗi kẻ thù sẽ có các thuộc tính riêng.

- Lớp EnemyManager là lớp quản lý kẻ thù, vì trong mỗi map thường có nhiều kẻ thù nên chúng ta cần quản lý list các kẻ thù trong map.

* Package Menu: Đâu là package quản lý các trạng thái game: bao gồm các trạng thái Continue(đang chơi), Menu(Màn hình chính),GameOption, QuitGame.



- Trong lớp Continue: Có các thuộc tính Nhân vậy, Quản lý kẻ thù, các trạng thái của game, các hàm bắt và xử lý sự kiện được truyền vào từ lớp gamePanel, có phương thức update trạng thái cho game, render…

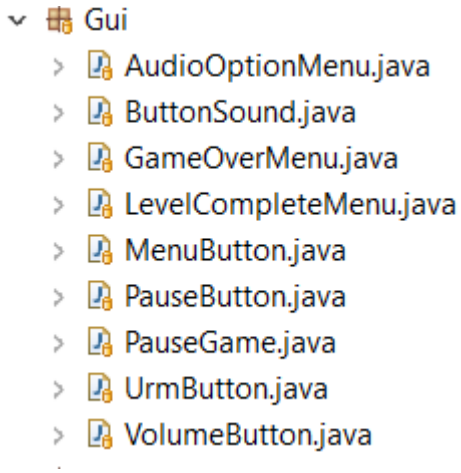
- Trong lớp GameStatus đây là lớp được đinh nghĩa là Enum: bao gồm các trạng thái của game CONTINUE, MENU, OPTIONS, QUIT;

- Trong lớp Status: Đây là lớp để set các trạng thái của game khi game chuyển trạng thái từ trạng thái này sang trạng thái khác thông qua sự thay đổi logic của các biến trạng thái của game. Ví dụ trạng thái của Game đang là Continue. Khi đó có một biến Paused là biến true/ false để cho biến trạng thái của game đang là gì. Nếu game đang chạy thì biến này sẽ được mặc định là false. Khi ta thực hiện cho game tạm dừng bằng cách truyền vào sự kiện nhấn 1 phím xuống, sau khi nhân phím xuống thì phím sẽ truyền vào keyInput, khi đó trình quản lý GamePanel sẽ set lại trạng thái biến Paused= true; biến Paused bằng true -> trạng thái Pause game sẽ được thiết lập cho

game, và màn hình Paused sẽ hiện lên. Nếu chúng ta nhấn nút tiếp tục trong màn hình Pause thì khi đó sự kiện kick vào button sẽ được thực hiện và biến paused sẽ đước set thành false, trạng thái continue sẽ được thiết lập và game lại quay trở lại trạng thái tiếp tục chơi. Có thể nói đây là cách thức mà các menu hoạt động cũng như cách thức di truyền của nhân vật trong game.

- Lớp StatusMethod: đây là interface dùng để định nghĩa các phương thức trừu tượng để ghi đè lại trong các trạng thái game. Các lớp continue, menu, gameOption sẽ kế thừa các interface này.

* Package GUI: Đây là package quản lý các giao diện cho các menu trạng thái game, như pause game, gameOption, menu chính…. Trong package có các lớp giúp quản lý, khởi tạo giao diện, các button.



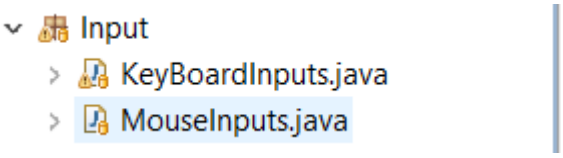
- Lớp Button Sound, volume, urmbutton, pausebutton: đều là các loại button được dùng để tạo các loại nút khác nhau để sử dụng lại nhiều làm. Trong các nút này đề có các MouseListener để lắng nghe sự kiện tương tác chuột.

- Lớp AudiooptionMenu: Khởi tạo giao diện cho gameOption, trong đây có khởi tạo các button điều khiển trạng thái của âm thanh, điều kiển volume ở trên.

- PauseGame: Khởi tạo giao diện Pause cho game, cũng có khởi tạo các button điển khiển trạng thái game, trạng thái âm thanh

- LevelCompleteMenu: Khởi tạo giao diện hoàn thành màn sau khi kết thúc màn, ở đây cũng khởi tạo các button điều khiển

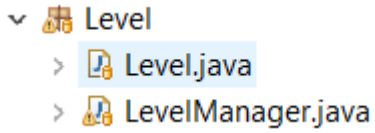
* Package Inputs: Quản lý các trình nhận sự kiện để xử lý sự kiện cho game như KeyListener, MouseListener



- KeyBoardInput: Lớp này để thiết lập các trình xử lý sự kiện khi ta tương tác với bàn phím

- MouseInput: lớp này để thiết lập các trình xử lý sự kiên khi ta tương tác với chuột

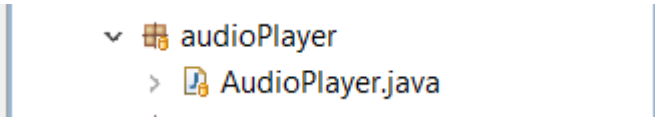
* Package Level: Đây là package để quản lý map



- Level: Quản lý từng level 1 bao gồn chỉ số định danh theo màu của các ô trong map, các ô định danh cho kẻ thù, định danh cho nhân vật, định danh cho item.

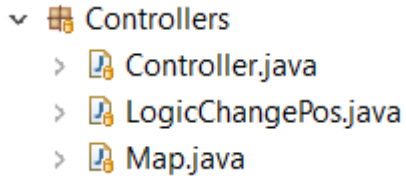
- Level Manager: Quản lý các level của game.

* Package audioPlayer: quan lý âm thanh cho game



- Lớp AudioPlayer: Lớp này dùng để quản lý các trình phát âm thanh cho game, thay đổi các hiệu ứng âm thanh, bật tắt âm thanh, điều khiển âm lượng.

* Package Controllers: chứa các lớp final tĩnh, các biến final tĩnh, các hàm check vị trí….



- Controller: Chứa các biến final tĩnh cho game, như là các trạng thái của nhân vật, của kẻ thù, các kích thước của các button, các hình menu…

- Map: Quản lý danh sách các bức ảnh sử dụng cho game. Có các phương thức đọc file ảnh

- LogicChangePos: Chứa các phương thức tĩnh dùng để kiểm tra vị trí nhân vật trong map, kiểm tra vị trí nhân vật có trướng ngoại vật hay không, nhân vật có đứng trong các ô có các chướng ngoại vật được vẽ lên map không, vị trí tiếp theo mà nhân vật di truyển có chướng ngoại vật hay không, khởi tạo định danh theo giá trị màu cho các ô trong map.

4. Giải thích các logic game

4.1 Logic di truyển của nhân vật và kẻ thù

Trước hết ta xét di truyển của nhân vật: nhân vật có các biến trạng boolean chỉ trạng thái di truyển, tấn công của nhân vật đó là left, right, jump, attack. Biến này mặc định là false. Khi ra thực hiện nhấn các nút trên bàn phím thì game sẽ có trình nhận sự

kiện sau khi nhận sự kiện xong thì nó sẽ thực hiện xử lý sự kiện, thay đổi tính logic của các biến trạng thái , các biến trạng thái chuyển về dạng true. Khi đó có một hàm cập nhật vị trí của nhân vật sẽ thực hiện kiểm tra điều kiện nếu như trạng thái left của nhân vật true thì sẽ thực hiện thay đổi tốc dộ di truyển của nhân vật. tương tự đối với các trạng thái khác bao gồm cả nhảy, và tấn công.

```
public void keyPressed(KeyEvent e) {
    if(gameOver)
        gameOverMenu.keyPress(e);
    else
        switch (e.getKeyCode()) {
        case KeyEvent.VK_A:
            player.setLeft(true);
            break;
        case KeyEvent.VK_D:
            player.setRight(true);
            break;
        case KeyEvent.VK_SPACE:
            player.setJump(true);
            break;
        case KeyEvent.VK_ESCAPE:// nếu keyListenner la space
            // của status menu sẽ được thực hiện
            paused = !paused;
            break;
        case KeyEvent.VK_J:
            player.setActtack(true);
            break;
        }
}
```

```
private void updatePosition() {
    moving = false;// hiện tại nhân vật không di truyển
    if (jump)// nếu nhân vật đang nhảy
        jump();// set trạng thái nhảy cho nhân vật như là tốc đ
    if (!left && !right && !inAir)// không thực hiện gì
        return;
    if(!inAir)
        if((!left&& !right)||(left&&right))
            return;
    float xSpeed = 0; // tốc đo hiện tại bằng 0
    if (left){// di truyển trái: tốc đô di truyển bằng tốc đô c
        xSpeed -=speed;
        flipX =width;// chiều rông thực tế của biến x là chiều
        flipW=-1;
    }
    if (right) {
        xSpeed +=speed;// di truyển phải: tốc đô di truyển của
        flipX=0;
        flipW=1;
    }
```

Tuy nhiên có một vấn đề là trong game thì nhân vật vẫn có thể di truyển ở khắp mọi nơi trên bản đồ bao gồm đi lên cả các chướng ngoại vật hay là cả không khí, như thế chúng ta cần phải trải map ra thành map dọc. Đó là thiết lập cho game một trọng

lực để nhân vật luôn rơi xuống dưới của map, cũng như là cần phải xét các vị trí mà nhân vật có thể đi tới.

Để thực hiện được điều này ta cần phải thêm cho nhân vật, cũng như kẻ thù các hộp bao quanh chứa toàn bộ nhân vật và kẻ thù. Các hộp này gọi là các hộp hitBox.

Để nhân vật có thể rơi trên map chúng ta cần phải xét xem dưới chân nhân vật có chướng ngoại vật nào không. Để xét như vậy thì chúng ta cần phải xét giao giữa hộp nhân vật và ô chứa nhân vật xem ô chứa nhân vật có chướng ngoại vật nào không. Như ta đã giải thích ở phần vẽ map thì mỗi ô trong map sẽ được định danh bởi một giá trị của giá trị màu. Ta cần biết được giá trị màu ở ô đó là bao nhiêu, nếu như giá trị màu ở ô đó nằm từ 0-48 và khác 11 thì ô đó có chướng ngoại vật tức là không thể đi được.

```
public static boolean checkPosTile(int xTile,int yTile,int [][] levelData)
    int value = levelData[(int) yTile][(int) xTile];// voi hai bien x, y t
    if(value >48 ||value <0|| value !=11) {// 11 chinh la sprite cua thang
        return true;
    }
    return false;
}
```

Đây chính là phương thức để kiểm tra xem ô đó có chướng ngoại vật hay không nếu giá trị là true tức là ô đó không có trướng ngoại vật false tức là có.

```
private static boolean checkPos(float x,float y, int [][] levelData) {//kiem tra xem
    int maxWidth = levelData[0].length*Game.TILES_SIZE;// chiều rộng max của map;
    if(x<0 || x>=maxWidth) {// check xem hiện tại nhân vật có thể đi được ở vị trí c
        return true;// nghia la hien tai o day dang co doi tuong chong cheo vao nhar
    }
    if(y<0||y>= Game.HEIGHT_SIZE) {
        return true;
    }
    float xIndex=x/Game.TILES_SIZE;//tao 1 bien luu toa do cua x
    float yIndex=y/Game.TILES_SIZE;//tao 1 bien luu toa do cua y
    return checkPosTile((int) xIndex, (int) yIndex, levelData);
}
```

Đây là phương thức để lấy ô đang đứng xong gọi đến hàm kiểm tra chướng ngoại vật của ô hiện tại

Tiếp theo chúng ta cần kiểm tra xem nhận vật có đang đứng trên chướng ngoại vật không nếu không thì tiếp tục rơi xuống nếu có thì cập nhật vị trí nhân vật.

```
if (!inAir)// nếu không ở trong không khí
    if (!IsEntityOnFloor(hitBox, levelData))// check chướng ngại vật ở ở trên đường đi
        inAir = true;

if (inAir) {// nếu ở trong không khí
    if (canMoveHere(hitBox.x, hitBox.y + airSpeed, hitBox.width, hitBox.height, levelData)) {// ch
        hitBox.y += airSpeed;// toạ độ của hộp sẽ thay đổi đến đó
        airSpeed += GRAVITY; // tốc độ trong không khí bằng trọng lực
        updateXPos(xSpeed);// thay đổi toạ độ x
    } else {// nếu có chướng ngoại vật
        hitBox.y = GetEntityYPosUnderRoofOrAboveFloor(hitBox, airSpeed);// lấy toạ độ của chướng n
        if (airSpeed > 0) // có chướng ngoại vật mà đang chuyển động thì sẽ dừng lại
            resetInAir();// đã rơi hết
        else//có chướng ngoại vật mà không chuyển động rơi xuống
            airSpeed = fallSpeedAfterCollision; // tiếp tục rơi
        updateXPos(xSpeed);// cập nhập vị trí x
    }
```

Tiếp theo chúng ta cần xét kiểm tra xem các vị trí mà nhân vật có thể di truyển trong map. Ta cũng cần xét xem ô tiếp theo mà nhân vật di truyển đến có chướng ngoại vật không nếu có thì không được di truyển nếu không thì tiếp tục được di truyển

```
public static boolean canMoveHere(float x,float y,float width,float height, int [][] levelData) {// ki
    if(!checkPos(x, y, levelData))// neu sai thi tiep tuc check, nghia la co the di tiep// check vi tr
        if(!checkPos(x+width, y+height, levelData))//check vi tri tren ben phai
            if(!checkPos(x+width, y, levelData))// check vi tri duoi
                if(!checkPos(x, y+height, levelData))// check vi tri duoi ben phai
                    return true;// neu dung thi duoc di tiep
    return false;//sai khong duoc di
    // tuy nhien van co the di chuyen qua nhung o nho neu ta chua bao gio va cham vao bat ki goc nao b
    // tai sao?

}
```

Trên đây là hàm kiểm tra xem vị trị tiếp theo nhân vật có di truyển được không

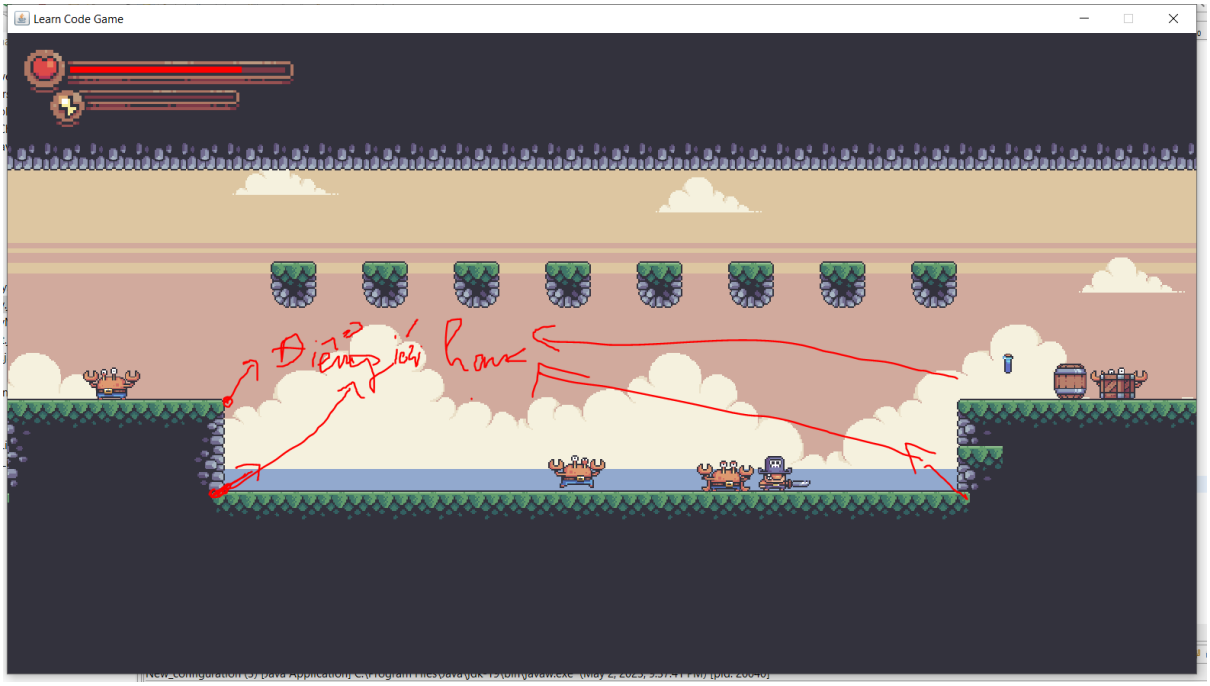
```
private void updateXPos(float xSpeed) {
    if (canMoveHere(hitBox.x + xSpeed, hitBox.y, hitBox.width, hitBox.height, levelData)) {// kiểm tra
        hitBox.x += xSpeed; // nếu được thì cập nhật
        System.out.println("x="+x);
    } else {
        hitBox.x = GetEntityXPosNextToWall(hitBox, xSpeed);// nếu không thấy vị trí của thằng cản trở
    }

}
```

* Logic di truyển của kẻ thù

Đối với kẻ thù cũng có các logic di truyển giống như nhân vật. Tuy nhiên chúng ta không thiết lập các điều khiển logic di truyển của kẻ thù như nhân vật mà logic di truyển của kẻ thù được tạo ra lặp đi lặp lại, không chịu sự điều khiển của người chơi.

Ta sẽ code một số hàm thiết lập di truyển cho kẻ thù:

- Khi kẻ thù chưa nhìn thấy nhân vật thì sẽ di truyển theo một hướng cho đến khi đi đến điểm giới hạn trái hoặc phải của map(điểm này là điểm mà tại đó có chướng ngoại vật)



Khi kẻ thù đi đến điểm giới hạn thì biến hướng đi của kẻ thù sẽ thay đổi

```
        if(walkDirection==LEFT)
            xSpeed=-speed;
        else
            xSpeed=speed;
        if(canMoveHere(hitBox.x+xSpeed, hitBox.y, hitBox.width, hitBox.height, levelData))
            if(IsFloor(hitBox,xSpeed, levelData)) {
                hitBox.x+=xSpeed;
                return;
            }
        changeWalkDirection();
        break;
    }
}
```

```
protected void changeWalkDirection() {
    if(walkDirection==LEFT) {
        walkDirection=RIGHT;
    }
    else
        walkDirection=LEFT;

}
```

Tiếp theo chúng ta cần phải thiết lập các logic di truyển cho kẻ thù khi có nhân vật

- Trước tiên ta cần phải kiểm tra xem nhân vật có nằm trong tầm ngắm của kẻ thù không nếu nhân vật đang nằm trong tầm ngắm của kẻ thù thì kẻ thù sẽ luôn đuổi theo nhân vật.

```
private boolean isPlayerInRagne(Player player) {
    int distance= (int) Math.abs(hitBox.x-player.hitBox.x);
    return distance<=acttackDistaince*5;
}
```

Khoảng cách kẻ thù và nhân vật

```
protected boolean canSeePlayer(int [][] levelData,Player player) {
    int playerTileY = (int) (player.getHitBox().y/Game.TILES_SIZE);//lấy toạ độ Y ô đang chứa nhân vật;

    if(playerTileY==enemyTileY)
        if(isPlayerInRagne(player))// player đang ở đủ gần kẻ thù
            if(IsSightClear(levelData,hitBox,player.hitBox,enemyTileY))// kiểm tra xem giữa nhân vật và kẻ
                return true;
    return false;
}
```

Check kẻ thù đang ở gần nhân vật

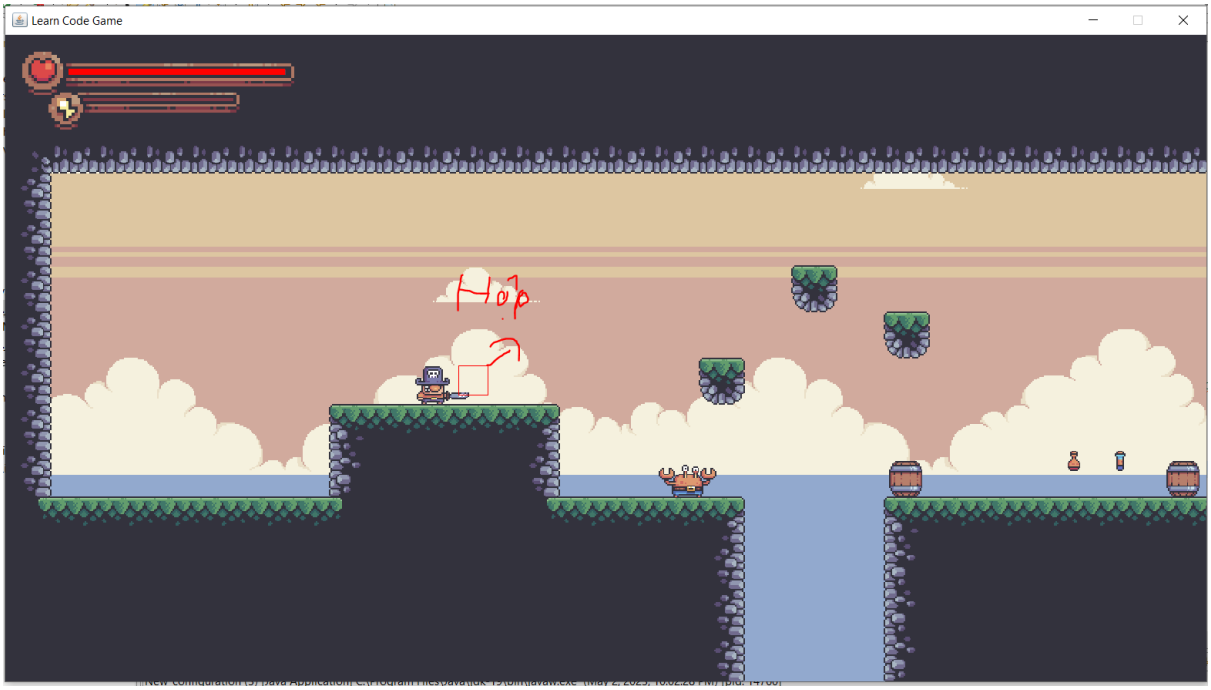
```
protected void turnTowardsPlayer(Player player) {// điều chỉnh di truyền của kẻ thù so với người chơi
    if(player.hitBox.x>hitBox.x)// người chơi đã đi ra sau kẻ thù
        walkDirection=RIGHT;
    else // người chơi đi ra trước kẻ thù
        walkDirection=LEFT;
}
```

Kẻ thù đuổi theo nhân vật

4.2. Logic gây sát thương của nhân vật và kẻ thù

* Nhân vật gây sát thương cho kẻ thù

Ta tạo cho nhân vật một vùng, vùng này có tác dụng xét va chạm với kẻ thù



Khi kẻ thù đi vào vùng không gian của hộp và khi đó trạng thái của nhân vật là attack thì kẻ thù sẽ bị trừ máu.

```
public void checkPlayerHit(Rectangle2D.Float attackBox) {
    for(Crabby c:crabbies)
        if(c.getCurrentHealth()>0)
            if(c.isAcitve())
            {
                if(attackBox.intersects(c.getHitBox())) {// kiểm t
                    c.hurt(5);
                    return;
                }
            }
}

private void checkActtack() {
    if( acttackChecked || statusIndex!=1)
        return;
    acttackChecked=true;
    continuee.checkEnemyDamage(acttackBox);
    continuee.checkItemHit(acttackBox);
    continuee.getGame().getAudioPlayer().playAttackSound();

}
```

* Kẻ thù gây sát thương cho nhân vật:

Như mục trên em đã đề cập đến quá trình di truyển của kẻ thù khi nhân vật đi vào tầm ngắm của nhân vật.

Tiếp theo em sẽ mô tả lại quá trình tấn công của kẻ thù

Trước tiên ta sẽ kiểm tra xem nhân vật có đang trong phạm vi tấn công của kẻ thù không

```
protected boolean isPlayerCloseForActtack(Player player) {// kiểm tra x
    int distance= (int) Math.abs(hitBox.x-player.hitBox.x);
    return distance<=acttackDistaince;
}
```

Tiếp theo ta sẽ kiểm tra va chạm giữa hộp tấn công của kẻ thù và hộp nhân vật nếu 2 hộp này giao nhau thì máu kẻ thù bị giảm

```
protected void checkEnemyHit(Rectangle2D.Float acttackBox, Player player) {// kiểm tra quái đánh
    if(acttackBox.intersects(player.hitBox))
        player.changeHealth(GetEnemyDamage(enemyType));
    acttackChecked =true;

}
```

Như vậy tổng kết phần 4.2 em đã giải thích về logic di truyển và logic tấn công của kẻ thù và nhân vật. Để đảm bảo dung lượng bài báo cáo nên em đã tóm lược ngắn gọn phần giải thích ngắn nhất có thể.

4.3. Logic chuyển trạng thái của game

Trong lớp Continue tức là lớp quản lý tất cả mọi đối tượng hiện thị hiện tại trong game. Trong lớp này sẽ có các biến boolean để chuyển các trạng thái menu cho game

```
private PauseGame pauseGame;
private GameOverMenu gameOverMenu;
private LevelCompleteMenu levelCompleteMenu;
private boolean paused = false ;// biến này cho biết khi nào sẽ ch
private boolean levelCompleted;
private boolean playerDying;
```

Khi game đang chạy thì biện pause sẽ là false. Khi ta thực hiện nhận phím tạm dừng cho game truyền vào phím escape ta đã thiết lập xử lý sự kiện khi nhấn phím escape khi nhấn vào thì pause sẽ chuyển thành true, trong lớp Continue có một hàm check điều kiện nếu pause =true thì sẽ truyển trạng thái của game đang tự status continue thành pause game, khi đó menu pause game sẽ hiện lên. Và tất cả các trạng thái của continue sẽ dừng hết lại

```
case KeyEvent.VK_ESCAPE:// nếu keyListe
    // của status menu sẽ được thực hiệ
    paused = !paused;
    break;
```

```
if(paused) {

    g.setColor(new Color(0, 0, 0, 150));
    g.fillRect(0, 0, Game.WIDTH_SIZE, Game.HEIGHT_SIZE);
    pauseGame.draw(g);
}
```

Khi ta nhấn tiếp tục chơi thì thiết lập lại giá trị false cho biến pause sau đó check điều kiền pause trên nếu không đúng thì sẽ không thiết lập trạng thái Pause nữa và quay về trạng thái continue.

```
else if(isIn(e,unPauseB)) {
    if(unPauseB.isMousePressed()) {// đang trong vị trí của musibutton
        continuee.unPauseGame();;// set chuyển trạng thái cho musicButton
    }
}
```

Đối với các trạng thái khác logic cũng tương tự như Pause Game.

Kết luận phần 4: Sau phần 4 em cũng đã tóm lược lại logic của game từ cách nhân vật di truyền, các nhân vật và kẻ thù tấn công, cách chuyển qua lại giữa các menu trạng thái game.

5. Kết Luận

Sau quá trình tự tìm hiểu trên internet, kết hợp với kiến thức thầy truyền đạt trên lớp em đã code ra được game cho bản thân. Em đã hiểu thêm về quá trình xây dựng các lớp cho một project, hiểu về cách hoạt động, các bước để phát triển ra một game. Sau này nếu có điều kiện về thời gian chắc chắc em sẽ tiếp tục hoàn thiện sản phẩm cũng như là có thêm một số game khác. Trên đây là báo cáo về bài tập lớn của em. Nếu còn nhiều thiếu sót, em rất mong nhận được sự góp ý của thầy để sản phẩm của em thêm hoàn thiện cũng như là em nhận được thêm những bài học về các dự án lần sau. Em trân thành cảm ơn thầy trong suốt học phần đã truyền đạt nhiều kiến thức cho em.

6. Tài liệu tham khảo

Toàn bộ slide của thầy.

https://uet.vnu.edu.vn/~chauttm/e-books/OOP_2013.pdf

https://chat.openai.com/

https://www.youtube.com/watch?v=6_N8QZ47toY&list=PL4rzdwizLaxYmltJQRjq18a9gsSyEQQ-0