HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO BÀI TẬP LỚN

MÔN HỌC: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Tên đề tài: Game Tank Battle

Giảng viên: Nguyễn Mạnh Sơn
Nhóm môn học: 02
Nhóm BTL: 16
Thành viên:
B21DCCN344 – Nguyễn Tiến Hiệp
B21DCCN414 – Nguyễn Sinh Hùng
B21DCCN721 – Đào Xuân Trí
B21DCCN750 – Nguyễn Huy Tú
B21DCCN759 – Nguyễn Minh Tuấn

Hà Nội, 30 tháng 11 năm 2023

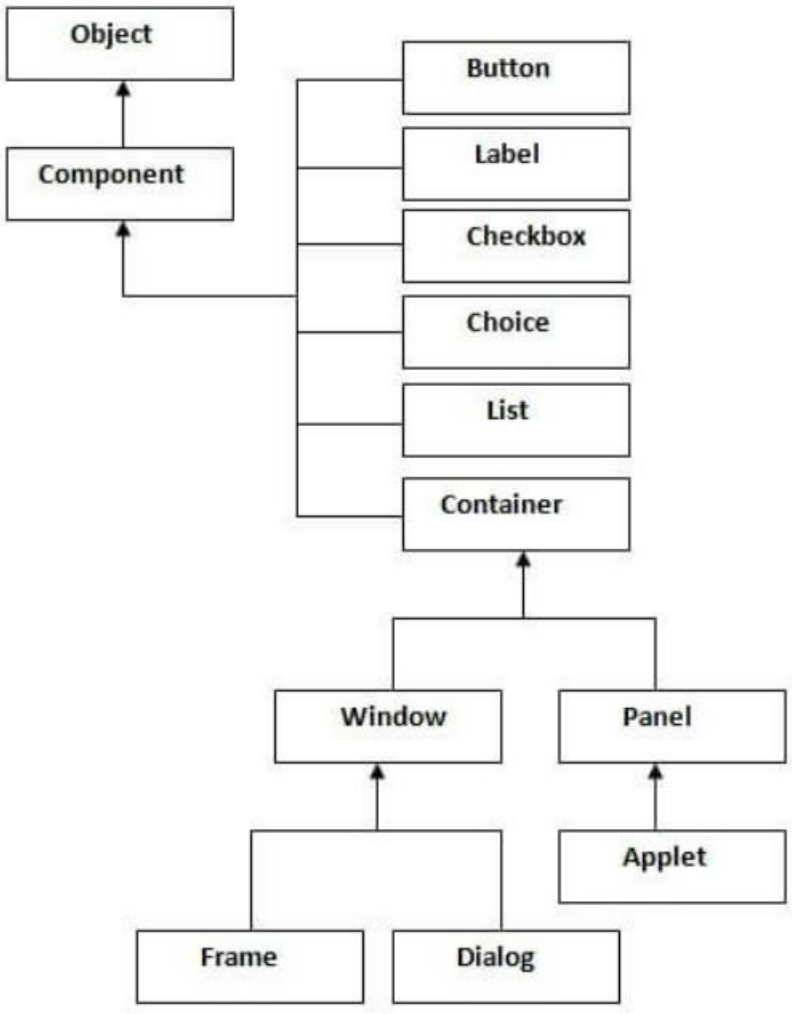
Mục lục

A. Các gói, thư viện được sử dụng trong game

I. Java AWT

Java AWT là một API để phát triển Giao diện người dùng đồ họa (GUI) hoặc các ứng dụng dựa trên cửa sổ trong Java. Gói java.awt cung cấp class cho AWT API như TextField, Label , TextArea, RadioButton, CheckBox, Choice, List….cung cấp các thành phần như button(nút), trường văn bản(text fields), thanh cuộn(scroll), v.v...

Hệ thống phân cấp của các lớp Java AWT được đưa ra dưới đây:



Container: Về cơ bản, nó là một màn hình nơi các thành phần được đặt tại các vị trí cụ thể của chúng. Vì vậy nó chứa và kiểm soát cách bố trí của các thành phần. Các lớp mở rộng của Container(vùng chứa): Khung(Frame), Hộp thoại(Dialog) và Bảng điều khiển(Panel), Window.

+Window: là vùng chứa không có đường viền và thanh menu. Bạn phải sử dụng khung, hộp thoại hoặc cửa sổ khác để tạo cửa sổ. Chúng ta cần lớp Window để tạo vùng chứa này.

+Panel: Bảng điều khiển là vùng chứa không chứa thanh tiêu đề, đường viền hoặc thanh menu. Nó là nơi chứa chung để chứa các thành phần như button, text, v.v. Lớp Panel tạo ra một vùng chứa, trong đó chúng ta có thể thêm các thành phần khác.

+Frame: Khung là vùng chứa thanh tiêu đề và đường viền và có thể có các thanh menu. Nó có thể có các thành phần khác như nút, trường văn bản, thanh cuộn, v.v. Khung là vùng chứa được sử dụng rộng rãi nhất khi phát triển ứng dụng AWT.

1. Event và Listener

Java cung cấp gói java.awt.event để xử lý các sự kiện trong giao diện người dùng, bao gồm chuột, bàn phím. Việc thay đổi trạng thái của mỗi đối tượng được gọi là một sự kiện. Ví dụ: nhấp vào nút, kéo chuột,... Gói java.awt.event cung cấp nhiều lớp sự kiện và giao diện Listener để xử lý sự kiện.

Các lớp Event và interface Listener tương ứng

Event Classes	Listener Interfaces	Description
KeyEvent	KeyListener	Lớp KeyEvent đại diện cho một sự kiện phím, bao gồm các hành động như phím được nhấn và phím được nhả ra.
MouseEvent	MouseListener and MouseMotionListener	Lớp MouseEvent đại diện cho một sự kiện chuột, bao gồm các hành động như nhấn chuột, nhả chuột và di chuyển chuột.

WindowEvent	WindowListener	Lớp WindowEvent: Đại diện cho một sự kiện liên quan đến cửa sổ, chẳng hạn như thay đổi kích thước, di chuyển hoặc đóng cửa sổ.

a) **Java KeyListener**

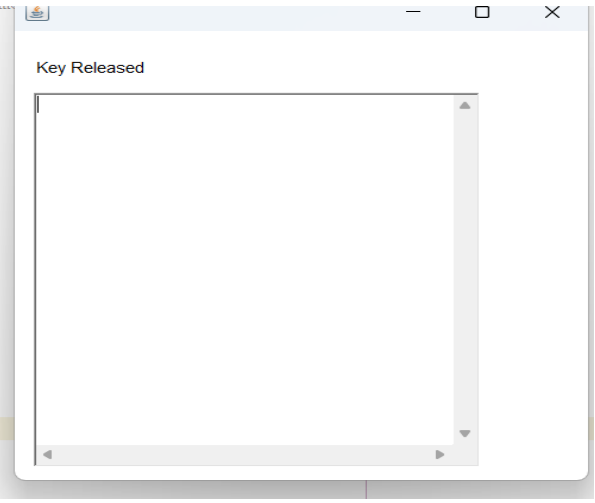
Java KeyListener được thông báo bất cứ khi nào bạn thay đổi trạng thái của bàn phím. Nó được thông báo đối với KeyEvent. Giao diện KeyListener được tìm thấy trong gói java.awt.event.

Khai báo Interface: java.awt.event.KeyListener: public interface KeyListener extends EventListener.

Các phương thức của KeyListener interface:

Method name	Description
public abstract void keyPressed (KeyEvent e);	Sự kiện này xảy ra khi một phím được nhấn xuống. Nó diễn ra ngay khi bạn bắt đầu nhấn một phím và không cần chờ đến khi bạn nhả phím. Do đó, nếu bạn giữ một phím xuống, sự kiện này sẽ được kích hoạt liên tục.
public abstract void keyReleased (KeyEvent e);	Khác với keyPressed, sự kiện này chỉ được thực hiện 1 lần duy nhất sau khi ta nhả phím ra.
public abstract void keyTyped (KeyEvent e);	Nó được gọi khi một phím được gõ.

```
        setSize (width: 400, height:400);
        setLayout (mgr: null);
        setVisible (b: true);
    }
/ overriding the keyPressed() khi nhấn key
    public void keyPressed (KeyEvent e) {
        l.setText (text: "Key Pressed");
    }
/ overriding the keyReleased() khi key được xóa
    public void keyReleased (KeyEvent e) {
        l.setText (text: "Key Released");
    }
/ overriding the keyTyped() khi gõ key
    public void keyTyped (KeyEvent e) {
        l.setText (text: "Key Typed");
    }
 // main method
    public static void main(String[] args) {
        new KeyListenerExample();
    }
```



b) Java MouseListener Interface

Java MouseListener được thông báo bất cứ khi nào bạn thay đổi trạng thái chuột, sử dụng để lắng nghe và xử lý các sự kiện chuột. Giao diện này khai báo các phương thức để xử lý các sự kiện chuột như khi chuột được nhấn, chuột được nhả ra, và khi chuột được nhấn và kéo. Nó được thông báo phản hồi **MouseEvent**. Giao diện MouseListener được tìm thấy trong gói java.awt.event.

Các phương thức của MouseListener:

+public abstract void mouseClicked(MouseEvent e); //Được gọi khi một nút chuột được nhấn và nhả ra ngay sau 1 khoảng thời gian ngắn và vị trí chuột không di chuyển quá nhiều.

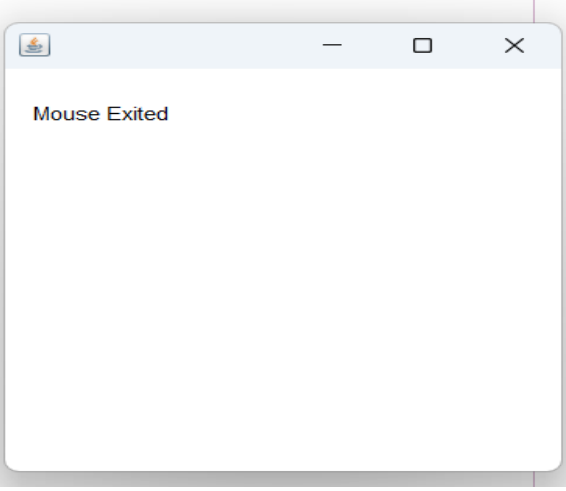
+public abstract void mouseEntered(MouseEvent e);//Được gọi khi con trỏ chuột đi vào vùng lắng nghe sự kiện (component).

+public abstract void mouseExited(MouseEvent e);//Được gọi khi con trỏ chuột rời khỏi vùng lắng nghe sự kiện.

+public abstract void mousePressed(MouseEvent e);//Được gọi khi một nút chuột được nhấn.. Nếu người dùng giữ nút chuột, sự kiện này sẽ liên tục được kích hoạt.

+public abstract void mouseReleased(MouseEvent e);//Được gọi khi một nút chuột được nhả ra.

```
        setVisible(b: true);
    }
    public void mouseClicked(MouseEvent e) {
        l.setText(text: "Mouse Clicked");
    }
    public void mouseEntered(MouseEvent e) {
        l.setText(text: "Mouse Entered");
    }
    public void mouseExited(MouseEvent e) {
        l.setText(text: "Mouse Exited");
    }
    public void mousePressed(MouseEvent e) {
        l.setText(text: "Mouse Pressed");
    }
    public void mouseReleased(MouseEvent e) {
        l.setText(text: "Mouse Released");
    }
ublic static void main(String[] args) {
```



c) Java MouseMotionListener Interface

Java MouseMotionListener được thông báo bất cứ khi nào bạn di chuyển hoặc kéo chuột. Nó được thông báo phản hồi MouseEvent. Giao diện MouseMotionListener được tìm thấy trong gói java.awt.event.

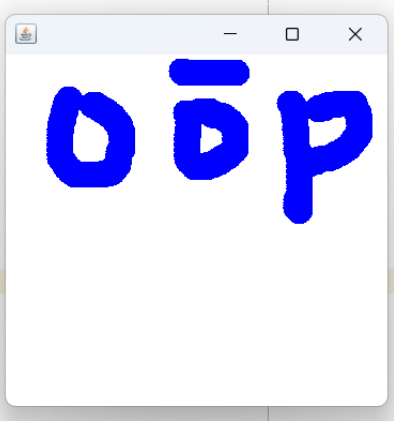
Các phương thức của giao diện MouseMotionListener

+public abstract void mouseDragged(MouseEvent e); //Phương thức này được gọi khi chuột di chuyển trên thành phần hoặc cửa sổ và đồng thời bấm và giữ nút chuột. Cũng giống như mouseMoved(), thông qua đối tượng MouseEvent, bạn có thể truy cập thông tin về vị trí hiện tại của chuột và các thuộc tính khác.

+public abstract void mouseMoved(MouseEvent e); //Phương thức này được gọi khi chuột di chuyển trên thành phần hoặc cửa sổ, nhưng không bấm nút chuột.

Ví dụ:

```
        setSize(width: 300,height:300);
        setLayout(mgr:null);
        setVisible(b: true);
    }
    @Override
    public void mouseDragged(MouseEvent e) {
    Graphics g=getGraphics();
    g.setColor(c: Color.BLUE);
    g.fillOval(x: e.getX(),y: e.getY(),width: 20,height:20);
}
    @Override
    public void mouseMoved(MouseEvent e) {}

public static void main(String[] args) {
    new MouseMotionListenerExample();
}
}
```



2. Lớp BufferedImage

Lớp BufferedImage thuộc gói java.awt.image là một lớp được sử dụng để đại diện cho một hình ảnh được lưu trữ trong bộ nhớ, cung cấp các phương thức để thao tác trực tiếp với dữ liệu hình ảnh. Một BufferedImage chứa dữ liệu hình ảnh dưới dạng các pixel (đơn vị cơ bản của hình ảnh), cho phép bạn truy cập và chỉnh sửa các giá trị màu sắc của từng pixel.

Một số phương thức quan trọng trong lớp BufferedImage:

+BufferedImage(int width, int height, int imageType): Constructor để tạo một đối tượng BufferedImage với chiều rộng, chiều cao và loại hình ảnh đã cho.

+int getWidth(): Trả về chiều rộng của hình ảnh.

+int getHeight(): Trả về chiều cao của hình ảnh.

+int getRGB(int x, int y): Trả về giá trị màu sắc của pixel tại tọa độ (x, y).

+void setRGB(int x, int y, int rgb): Thiết lập giá trị màu sắc của pixel tại tọa độ (x, y).

+Graphics2D createGraphics(): Tạo và trả về một đối tượng Graphics2D để vẽ và thao tác với hình ảnh.

→**Lớp BufferedImage** cung cấp các phương thức khác nhau để truy xuất và chỉnh sửa thông tin về hình ảnh. Có thể sử dụng nó để vẽ các hình dạng, văn bản và hình ảnh trên một hình ảnh, xử lý và chỉnh sửa hình ảnh, cắt, kết hợp và chuyển đổi hình ảnh, và nhiều tác vụ khác liên quan đến xử lý hình ảnh.

Ví dụ: Trong project đã sử dụng BuferedImage đại diện cho 1 ảnh vào dùng 1 đối tượng lớp Graphics2D để in ra màn hình

```
// vẽ game
public void drawGame(Graphics2D g, ImageObserver io) {
    // vẽ nền
    BufferedImage image = ResourceGetter.getBufferedImage("groundTexture.jpg");

    // Do 50 pixel đầu là phần Level rồi nền vị trí tung độ y phải tăng theo 50
    // pixel
    g.drawImage(image, 0, 0, FRAME_WIDTH, FRAME_HEIGHT + 50, io);

    // vẽ trạng thái game hiện tại
    gameStates[currGameStateIndex].draw(g, io);

}
```

3. Lớp Graphics2D

Lớp Graphics2D thuộc gói java.awt, kế thừa từ lớp Graphics. Lớp cung cấp các phương thức bổ sung và chức năng mạnh mẽ hơn để vẽ đồ họa so với Graphics. Nó là một phần của Java 2D API và thường được sử dụng khi bạn làm việc với GUI trong ứng dụng Java.

Một số phương thức quan trọng của lớp Graphics2D

+ **drawLine(int x1,int y1,int x2,int y2)**: Vẽ đường thẳng từ (x1,y1) đến (x2,y2).

+ drawRect(int x, int y, int width, int height): Vẽ hình chữ nhật tại vị trí (x, y) với chiều rộng và chiều cao cho trước.

+ **drawString(String str,int x,int y):** Vẽ chuỗi văn bản str tại vị trí (x,y).

+ **setColor(Color c):** Đặt màu vẽ.

+ **drawImage(BufferedImage a, int x,int y,int w,int h,ImageObserver io):** In ảnh có kích thước w*h tại vị trí (x,y)

Ví dụ: Trong game sử dụng lớp Graphics2D để vẽ đạn

```
public void draw(Graphics2D g) {
    // khi vừa bắn ra khỏi tank, nó được vẽ như một "vụ nổ"
    if(life < START_KILL_FRAME){
        // kích thước "vụ nổ" mở rộng rồi co lại.
        // lấy thời gian giữa "trung tâm" của vụ nổ
        // và thời gian hiện tại để tính kích thước hiện tại của vụ nổ
        int currDiameter = 5 * (4 - (Math.abs(life - 4)));
        int currRadius = currDiameter / 2;

        // vẽ vòng ngoài của vụ nổ màu đỏ
        g.setColor(Color.RED);
        g.fillOval((int)getX() - currDiameter, (int)getY() - currDiameter, currDiameter * 2, currDiameter * 2);
        // vẽ vòng trong của vụ nổ màu vàng
        g.setColor(Color.YELLOW);
        g.fillOval((int)getX() - currRadius, (int)getY() - currRadius, currDiameter, currDiameter);
    } else if(LIFETIME - life < 8){
        // khi viên đạn sắp bị loại bỏ, nó cũng được vẽ như một vụ nổ
        int currDiameter = 5 * (4 - (Math.abs((LIFETIME - life) - 4)));
        int currRadius = currDiameter / 2;
        // vẽ vòng ngoài của vụ nổ
        g.setColor(Color.RED);
        g.fillOval((int)getX() - currDiameter, (int)getY() - currDiameter, currDiameter * 2, currDiameter * 2);
        // vẽ vòng trong của vụ nổ
        g.setColor(Color.YELLOW);
        g.fillOval((int)getX() - currRadius, (int)getY() - currRadius, currDiameter, currDiameter);
    }
    // nếu viên đạn không phải vừa được bắn ra hoặc sắp bị loại bỏ, vẽ nó như một viên đạn bình thường
    else{
        g.setColor(color);
        g.fillOval((int)getX() - RADIUS, (int)getY() - RADIUS, DIAMETER, DIAMETER);
    }
}
```

II. Javax.sound.sampled

Trong Java Gói javax.sound.sampled cung cấp các lớp và giao diện để làm việc với âm thanh trong môi trường Java. Nó cho phép bạn đọc, ghi, xử lý và phát các tệp âm thanh và dữ liệu âm thanh trong ứng dụng của mình.

1. Lớp AudioInputStream

Lớp **AudioInputStream** trong gói javax.sound.sampled đại diện cho một luồng dữ liệu âm thanh. Nó cung cấp các phương thức để đọc dữ liệu âm thanh từ các nguồn khác nhau.

2. Lớp AudioSystem

Lớp **AudioSystem** trong gói javax.sound.sampled cung cấp các phương thức tĩnh để tạo và quản lý các tài nguyên âm thanh dựa trên lớp AudioInputStream cung cấp. Dưới đây là một số phương thức quan trọng của lớp AudioSystem:

+**getAudioInputStream(File file):** Phương thức này trả về một đối tượng AudioInputStream từ một tệp tin âm thanh đã cho. Nó cho phép bạn đọc dữ liệu âm thanh từ tệp tin và sử dụng nó trong các hoạt động như phát lại và xử lý âm thanh.

+**getAudioInputStream(InputStream stream)**: Phương thức này trả về một đối tượng AudioInputStream từ một InputStream đã cho. Bạn có thể sử dụng phương

thức này để đọc dữ liệu âm thanh từ luồng dữ liệu thông qua InputStream và sử dụng nó cho các mục đích như phát lại và xử lý âm thanh.

+**getAudioInputStream(URL url):** Phương thức này trả về một đối tượng AudioInputStream từ một URL đại diện cho tệp tin âm thanh. Nó cho phép bạn đọc dữ liệu âm thanh từ URL và sử dụng nó trong các hoạt động như phát lại và xử lý âm thanh.

+**isFileTypeSupported(AudioFileFormat.Type fileType):** Phương thức này kiểm tra xem định dạng tệp tin âm thanh đã cho có được hỗ trợ bởi hệ thống không. Nó trả về true nếu định dạng được hỗ trợ và false nếu không.

+**getClip():** Phương thức này trả về một đối tượng Clip mới. Clip là một lớp đại diện cho một đoạn âm thanh có thể được chơi và điều khiển. Bạn có thể sử dụng getClip() để tạo một đối tượng Clip mới để phát lại âm thanh.

+**getSourceDataLine(AudioFormat format):** Phương thức này trả về một đối tượng SourceDataLine từ định dạng âm thanh đã cho. SourceDataLine là một lớp đại diện cho một dòng dữ liệu âm thanh đầu ra. Nó cho phép bạn ghi dữ liệu âm thanh vào và phát lại từ dòng dữ liệu âm thanh.

Ví dụ:

```
AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(getClass().getResource(soundFileName));
```

3. Lớp Clip

Lớp Clip trong gói javax.sound.sampled.* đại diện cho một đoạn âm thanh có thể được chơi và điều khiển. Nó cung cấp các phương thức để phát lại, tạm dừng, tiếp tục và điều khiển các thuộc tính liên quan đến âm thanh. Dưới đây là một số phương thức quan trọng của lớp Clip:

+void open(AudioFormat format, byte[] data, int offset, int bufferSize): Phương thức này mở một đoạn âm thanh với định dạng âm thanh, dữ liệu âm thanh đã cho và kích thước bộ đệm. Bạn có thể sử dụng phương thức này để tạo một đoạn âm thanh từ dữ liệu và định dạng đã có.

+void open(AudioInputStream stream): Phương thức này mở một đoạn âm thanh từ một AudioInputStream. Bạn có thể sử dụng phương thức này để tạo một đoạn âm thanh từ luồng dữ liệu âm thanh.

+void start(): Phương thức này bắt đầu phát lại đoạn âm thanh từ điểm bắt đầu. Nó chạy đồng thời với luồng hiện tại.

+void stop(): Phương thức này dừng phát lại đoạn âm thanh và đưa con trỏ phát lại về điểm bắt đầu.

+void close(): Phương thức này đóng đoạn âm thanh và giải phóng tài nguyên liên quan.

+void loop(int count): Phương thức này lặp lại phát lại đoạn âm thanh. Đối số count chỉ định số lần lặp lại. Sử dụng giá trị -1 để lặp lại vô số lần.

+int getFrameLength(): Phương thức này trả về số khung (frames) trong đoạn âm thanh. Một khung (frame) là một tập hợp các mẫu âm thanh, mỗi mẫu gồm một số lượng byte tương ứng với độ phân giải âm thanh.

+ void setFramePosition(int x): Phát âm thanh tại vị trí frame x.

Ví dụ:

```
    clip = AudioSystem.getClip();
    clip.open(audioInputStream);
```

```
public void play() {
    if (clip != null) {
        clip.setFramePosition(0);
        decreaseVolume(-30.0f);
        clip.start();
    }
}
```

4. Lớp BooleanControl

Lớp BooleanControl trong gói javax.sound.sampled.* đại diện cho một điều khiển dạng boolean trong hệ thống âm thanh. Nó cho phép bạn kiểm soát các thuộc tính có giá trị **true** hoặc **false** liên quan đến các tài nguyên âm thanh. lớp BooleanControl kế thừa các phương thức từ lớp Control.

+ **getType**()://đây phương thức trong bài tập lớn chúng em kế từ lớp Control//. Phương thức này trả về kiểu của điều khiển. Trong trường hợp BooleanControl, kiểu sẽ là **BooleanControl.Type.MUTE** hoặc **BooleanControl.Type.ONOFF**.

+ Kiểu **BooleanControl.Type.MUTE** được sử dụng khi điều khiển boolean điều chỉnh chức năng tắt tiếng (mute) âm thanh. Điều khiển này cho phép bạn bật hoặc tắt tiếng âm thanh.

+ Kiểu **BooleanControl.Type.ONOFF** được sử dụng khi điều khiển boolean điều chỉnh chức năng bật/tắt (on/off). Điều khiển này cho phép bạn bật hoặc tắt một tính năng hoặc chế độ cụ thể.

5. Lớp javax.sound.sampled.FloatControl

Lớp javax.sound.sampled.FloatControl là một lớp trong gói javax.sound.sampled và cung cấp các phương thức và thuộc tính để điều khiển các giá trị số thực trong hệ thống âm thanh.

- Dưới đây là 1 số phương thức:

  + getType()://đây là phương thức của lớp FloatControl mà chúng em sử dụng trong btl//. Phương thức này trả về kiểu của điều khiển.

  Ví dụ:

  + **FloatControl.Type.VOLUME** được sử dụng để điều khiển âm lượng,

  + **FloatControl.Type.PAN** được sử dụng để điều khiển phân bổ âm thanh,

  + **FloatControl.Type.MASTER_GAIN** bạn có thể điều chỉnh âm lượng chung của hệ thống âm thanh, ảnh hưởng đến tất cả các nguồn âm thanh đang phát. Thông số âm lượng chính (master gain) thường được đo bằng đơn vị decibel (dB).

  Ví dụ:

```
    //Tạo 1 đối tượng kiểu FloatControl để điều chỉnh âm thanh
    volumeControl=(FloatControl) clip.getControl(FloatControl.Type.MASTER_GAIN);
    }
// Phương thức giảm âm lượng (volume là giá trị âm lượng mới, ví dụ: -10.0f)
   public void decreaseVolume(float volume) {
       if (volumeControl != null) {
           float currentVolume = volumeControl.getValue();
           volumeControl.setValue(currentVolume + volume);
       }
   }
```

6. Lớp javax.sound.sampled.LineUnavailableException và javax.sound.sampled.UnsupportedAudioFileException

- Lớp javax.sound.sampled.LineUnavailableException là một lớp trong gói javax.sound.sampled và là một ngoại lệ (exception) được ném ra khi không thể mở hoặc truy cập được một dòng âm thanh (audio line).
- Lớp javax.sound.sampled.UnsupportedAudioFileException là một lớp trong gói javax.sound.sampled và là một ngoại lệ (exception) được ném ra khi một tệp âm thanh không được hỗ trợ hoặc không thể đọc được.
- Mặc dù cả hai đều là các ngoại lệ (exceptions) được sử dụng trong gói javax.sound.sampled, nhưng chúng có mục đích và tình huống sử dụng khác nhau.

  + **LineUnavailableException**: Được ném ra khi không thể mở hoặc truy cập một dòng âm thanh (audio line). Ví dụ, khi cố gắng mở một dòng ghi âm (input line) hoặc một dòng phát âm (output line) nhưng không khả dụng hoặc bị sử dụng bởi ứng dụng khác, ngoại lệ này sẽ được ném ra.

  + **UnsupportedAudioFileException:** Được ném ra khi một tệp âm thanh không được hỗ trợ hoặc không thể đọc được. Ví dụ, khi cố gắng đọc một tệp âm thanh với định dạng không được hỗ trợ hoặc tệp âm thanh bị hỏng, ngoại lệ này sẽ được ném ra.

  →Tóm lại, **LineUnavailableException** xảy ra khi có vấn đề với các dòng âm thanh (audio lines) như mở, truy cập hoặc sử dụng chúng, trong khi **UnsupportedAudioFileException** xảy ra khi có vấn đề với tệp âm thanh, chẳng hạn như định dạng không được hỗ trợ hoặc tệp bị hỏng.

B. Giới thiệu, khảo sát và các module chính trong Game

I. Giới thiệu

- Sản phẩn: Tank Battle
- Thể loại: Fighting game (Game chiến đấu)
- Lối chơi:

+ Game có tất cả 12 level, mỗi level sẽ có một bản đồ khác nhau với những nhân vật như: 1 xe tăng (của người chơi) và 1 vài xe tăng địch.

+ Xe tăng có tất cả 3 loại gồm: loại thường, loại nhanh, loại bắn đạn đặc biệt

+ Nhiệm vụ của người chơi là dùng các nút điều hướng để di chuyển và chuột để bắn 1 cách chính xác để giúp người chơi giành chiến thắng qua các level.

II. Khảo sát

- Qua tìm hiểu ban đầu, nhận thấy game là một sản phẩm không quá khó và nhóm bọn em có thể tìm hiểu và làm được.
- Về thể loại, game dễ tiếp cận nhiều người chơi, chủ yếu chơi để giải trí sau những giờ học căng thẳng. Mặc dù cách chơi game đơn giản nhưng để giành chiến thắng hết 12 level thì rất là khó. Chính vì điều này càng làm game hấp dẫn, khiến cho người chơi dành nhiều thời gian chơi hơn để phá đảo.
- Khảo sát trải nhiệm người dùng, mọi người đều đánh giá game như sau:

+ Đồ họa game 2D bắt mắt, chơi mượt mà, không giật lag.

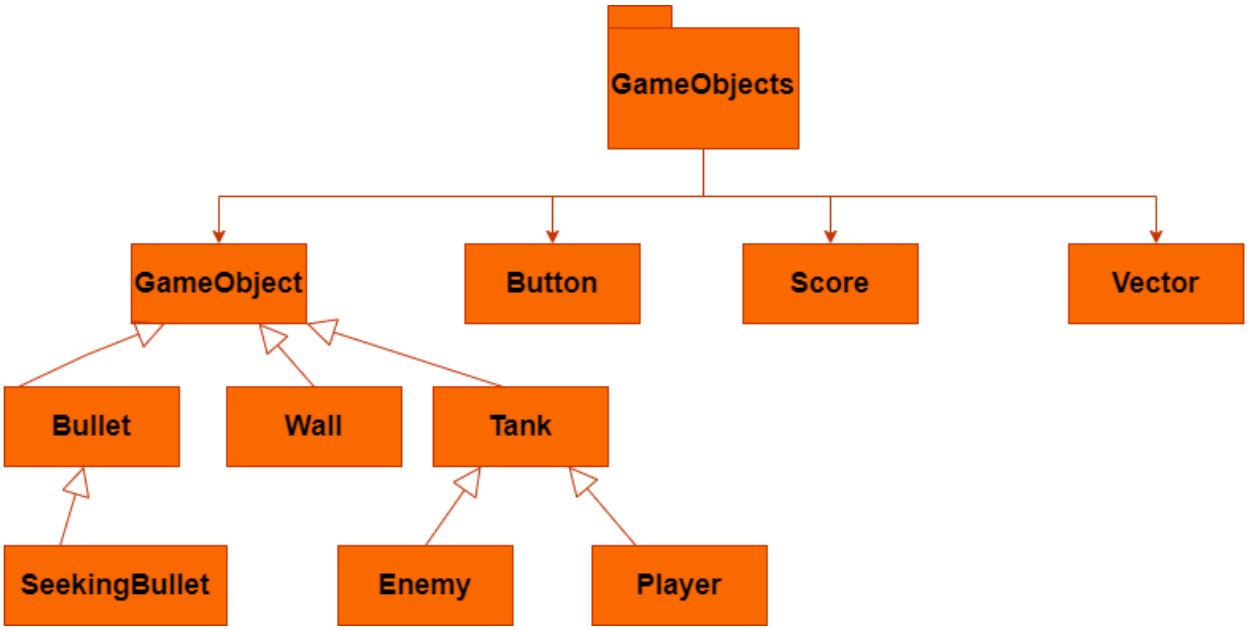
+ Cách chơi đơn giản, nhưng để phá đảo game thì rất khó.

+ Các level sắp sếp khó dần.

+ Âm thanh máu lửa.

III. Các module chính trong game

1. GameObjects



- Trong game, nhóm xác định có 6 đối tượng chính gồm: Tank (xe tăng), Bullet (đạn), Wall (tường), Button (nút bấm), Score (điểm số), Vector (tọa độ). Trong Bullet lại có class con SeekingBullet (đạn đặc biệt), Tank có 2 class con là Enemy (địch), Player (người chơi).
- Sau đây nhóm sẽ trình bày

a) Lớp Tank

- Các thuộc tính

```
//Sử dụng các số nguyên đại diện cho các kiểu xe tăng có trong game
final static public int NORMAL_TYPE = 0;
final static public int FAST_TYPE = 1;
final static public int SEEKING_TYPE = 2;

// các thông số kích thước hằng
public static final int BODY_WIDTH = 50;
public static final int BODY_HEIGHT = 50;
public static final int TURRET_LENGTH = 30;
public static final int TURRET_WIDTH = 5;
public static final int BODY_ROTATION_SPEED = 5;//tốc độ quay phần thân mỗi khung hình

//Các thuộc tính của đối tượng tank
protected float vX;
protected float vY;
protected float maxSpeed;
protected Color bodyColor;
protected Color turretColor;
protected int bodyAngle;
protected int turretAngle;
protected int type;
private int rotatedThisFrame = 0; //xe tăng đã xoay bao nhiêu độ
```

- Các phương thức chính:

  + Hàm khởi tạo: nhận vào giá trí x,y là tọa độ, angle là góc quay ban đầu của xe tăng và type là loại xe tăng.

```
public Tank(float x, float y, int angle, int type) {
    super(x,y);
    turretAngle = angle;
    bodyAngle = angle;
    bodyColor = Color.LIGHT_GRAY;
    turretColor = Color.BLACK;
    this.type = type;
}
```

  + Phương thức **draw(Graphics2D g):** được sử dụng để vẽ đối tượng xe tăng lên màn hình.

```
public void draw(Graphics2D g) {

    //Ép các giá trị tọa độ x,y về số nguyên
    int tankX = (int)getX();
    int tankY = (int)getY();
    int tankCenterX = (int)getCenterX();
    int tankCenterY = (int)getCenterY();


    //TANK BODY
    AffineTransform old = g.getTransform();
    g.rotate(Math.toRadians(bodyAngle), tankCenterX, tankCenterY);


    g.setColor(bodyColor);
    g.fillRect(tankX, tankY, BODY_WIDTH - 5, BODY_HEIGHT);
    g.fillPolygon(new int[]{tankX + BODY_WIDTH - 7, tankX + BODY_WIDTH - 7, tankX + BODY_WIDTH + 2}, new int[]{tankY, tankY + BODY_HEIGHT, tankY + BODY_HEIGHT/2}, 3);

    g.setColor(Color.BLACK);
    g.fillRect(tankX, tankY, BODY_WIDTH - 5, BODY_HEIGHT/6);
    g.fillRect(tankX, tankY + (int)Math.round(BODY_HEIGHT*(5.0/6.0)), BODY_WIDTH - 5, BODY_HEIGHT/6);
    g.setTransform(old);

    //TANK TURRET
    old = g.getTransform();
    g.rotate(Math.toRadians(turretAngle), tankCenterX, tankCenterY);

    g.setColor(turretColor);
    g.fillRect(tankCenterX - (TURRET_WIDTH)/2, tankCenterY - TURRET_WIDTH/2, TURRET_LENGTH, TURRET_WIDTH);
    g.fillRect(tankCenterX - 10, tankCenterY - 10, 20, 20);
    g.setTransform(old);

}
```

+ Phương thức **update():** được sử dụng để cập nhật trạng thái của xe tăng như: vị trí, vận tốc, góc quay.

```
public void update() {

    //Tính tốc độ trong trường hợp xe tăng không đi thẳng
    double totalSpeed = Math.sqrt(vX*vX + vY*vY);
    // Nếu tốc độ lớn hơn tốc độ tối đa thì phải thay đổi lại
    if(totalSpeed>maxSpeed){
        double ratio = (double)maxSpeed / totalSpeed;
        vX *= ratio;
        vY *= ratio;
    }

    //Cập nhật lại tọa độ
    setX(getX() + vX);
    setY(getY() + vY);

    //Chuẩn hóa lại góc có giá trị trong khoảng (0,360) độ
    if(bodyAngle < 0) bodyAngle += 360;
    if(bodyAngle >= 360) bodyAngle = bodyAngle%(360);

    if(turretAngle < 0) turretAngle += 360;
    if(turretAngle >= 360) turretAngle = turretAngle%(360);


    if(vX != 0 || vY != 0){
        rotateBodyToMovement();
    }

}
```

+ Ngoài ra còn 1 phương thức update() nữa được sử dụng để cập nhật trạng thái của xe tăng dựa trên các va chạm với tường, đạn, địch. Phương thức trả về một giá trị boolean cho biết xe tăng có bị hủy bỏ hay không

```
public boolean update(ArrayList<Wall> walls, ArrayList<Bullet> bullets, ArrayList<Enemy> enemies, Tank player){
    rotatedThisFrame = 0;
    this.update();

    resolveWallCollisions(walls);
    resolveTankCollisions(enemies, player);

    return !isCollidingBullet(bullets);
}
```

+ Phương thức isCollidingBullet(java.util.ArrayList<GameObjects.Bullet> bullets): Kiểm tra xem xe tăng có va chạm với bất kỳ đạn nào trong danh sách đạn được cung cấp hay không.

+ Phương thức resolveTankCollisions(ArrayList<Tank> walls) và resolveWallCollisions(ArrayList<Wall> tanks): Xử lý va chạm với các xe tăng và tường tương ứng.

+ Phương thức shootBullet(boolean imaginary): Bắn một viên đạn từ khẩu pháo của xe tăng. Tham số imaginary xác định xem đạn được bắn có thực sự tồn tại hay chỉ là ảo.

b) Player

- Lớp Player giống lớp Tank nhưng có thêm 1 số thuộc tính khác:

```
private static final float MAX_SPEED = 2;//pixels/frame

//2 biến đại diện cho vị trí của con trỏ chuột
private int mouseX;
private int mouseY;
//Các biến kiểu boolean có giá trị true nếu nút được nhấn, nếu không nhận false
private boolean leftPressed;
private boolean rightPressed;
private boolean upPressed;
private boolean downPressed;
//Số frame kể từ lần cuối người chơi bắn đạn
private int framesSinceShot;

private static final int COOLDOWN_TIME = 30;//số frame mà sau khi bắn đạn xong, người chơi phải chờ để được bắn tiếp
```

- Các phương thức draw(), update(), keyPressed, keyReleased(),mouseMoved(), shootBullet():

    + **draw(Graphics2D g):** để vẽ đối tượng người chơi lên màn hình.

    + **update():** cập nhật trạng thái của đối tượng người chơi.

    + **keyPressed(KeyEvent e):** xử lý sự kiện khi người dùng nhấn phím di chuyển.

+ **keyReleased(KeyEvent e):** xử lý sự kiện khi người dùng nhả phím ra.

+ **mouseMoved(int[] newPos):** được gọi khi con trỏ chuột di chuyển và xử lý sự kiện tương ứng. Trong trường hợp này, nó nhận vào một mảng newPos chứa tọa độ mới của con trỏ chuột.

+ **shootBullet():** được sử dụng để bắn một viên đạn từ người chơi. Nó trả về một đối tượng đạn (GameObjects.Bullet)

c) Lớp Enemy

- Lớp Enemy được khai báo là lớp con của lớp GameObject.Tank, lớp Enemy định nghĩa các thuộc tính và phương thức để quản lý và cập nhật các đối tượng kẻ địch trong trò chơi. Các phương thức này bao gồm việc vẽ kẻ địch, cập nhật trạng thái, tính toán lại đường đi, tạo bản sao và kiểm tra đường đi rõ ràng. Lớp Enemy có các thuộc tính sau :

```
private int timeUntilShoot;//khoảng thời gian cho tới khi địch bắn tiếp đạn
final private static int SHOOT_DELAY = 300;// số khung hình delay giữa mỗi lần bắn (mặc định là 300)
final private static float MAX_SPEED = (float)1.25;//tốc độ tối đa
```

- Các phương thức chính của Enemy

+ Hàm khởi tạo: Chúng ta sẽ khởi tạo Enemy cùng với các tham số tọa độ, góc quay ban đầu.

```
public Enemy(float x, float y, int angle, int type) {

    super(x, y, angle, type);
    // Mỗi xe tăng có kiểu đạn khác nhau sẽ có màu khác nhau để dễ phân biệt
    if(type == Tank.FAST_TYPE){
        this.bodyColor = new Color(70, 70, 210);
    }else if(type == Tank.SEEKING_TYPE){
        this.bodyColor = Color.RED;
    }else{
        this.bodyColor = Color.YELLOW;
    }
    this.turretColor = bodyColor.darker();
    timeUntilShoot = (int)(Math.random()*SHOOT_DELAY);
    this.maxSpeed = MAX_SPEED;
}
```

+ **draw():** vẽ đối tượng kẻ địch lên màn hình

+ **update():** phương thức này cập nhật trạng thái của kẻ địch.

+ **recalculatePath():** tính toán đường đi cho kẻ địch dựa vào trạng thái hiện tại của trò chơi.

+ **hasClearPath():** kiểm tra xem có đường đi rõ ràng(không có chướng ngại vật) giữa kẻ địch và người chơi.

- Để xe tăng địch di chuyển sao cho tối ưu nhất (nghĩa là di chuyển để nhanh đến người chơi nhất) tránh tình trạng di chuyển lung tung, nhóm đã áp dụng thuật toán tìm kiếm theo chiều rộng (BFS) được trình bày trong lớp findPathing(). Mọi người có thể tham khảo. Nhóm xin tóm tắt qua thuật toán gồm 3 bước sau:
  + Bước 1: Tạo 1 hàng đợi. Đưa vị trí hiện tại vào hàng đợi.
  + Bước 2: Lấy vị trí đầu hàng đợi ra khỏi hàng đợi. Đưa các vị trí xung quanh mà chưa đi qua vào cuối hàng đợi.
  + Bước 3: Lặp lại bước 2 đến khi nào gặp được điểm đích thì dừng. Nếu hàng đợi trống nghĩa là không tìm được đường đi đến điểm đích.

d) Lớp Bullet

- Class này được thiết kế để quản lý và điều khiển các đạn trong trò chơi. Nó bao gồm các thuộc tính để xác định vị trí, tốc độ và hướng di chuyển của đạn, cũng như các phương thức để vẽ đạn lên màn hình và cập nhật vị trí của đạn trong game. Nó cũng xử lý va chạm với các vật thể khác như tường và kiểm tra thời gian sống của đạn để xóa nó khi đạt đến giới hạn thời gian. Ngoài ra, lớp này có các thuộc tính để xác định loại đạn (nhanh, ảo) và thay đổi hành vi di chuyển tương ứng. Điều này giúp tạo ra đa dạng trong cách đạn được tạo ra và di chuyển trong trò chơi.

- Các thuộc tính chính:

```
//final class variables (constants)
final static public float DEFAULT_SPEED = 3; // điểm ảnh / khung
final static public float DEFAULT_FAST_SPEED = (float) 4.8; // điểm ảnh / khung. Tốc độ của viên đạn nhanh (một số kẻ địch bắn nhanh)
final static public int DIAMETER = 10;
final static public int RADIUS = DIAMETER / 2;
// Khung hình cho đến khi viên đạn có thể giết được một thứ gì đó (để tránh nhân vật giết chính mình khi bắn ra một viên đạn)
final static public int START_KILL_FRAME = 8;
// Số khung hình mà viên đạn tồn tại cho đến khi nổ.
final private static int LIFETIME = 300;

private float vX; // vận tốc x
private float vY; // vận tốc y
private int angle; // góc, theo độ, hướng của vận tốc viên đạn
private int life; // số khung hình(thời gian tồn tại) kể từ viên đạn được tạo (được sử dụng cho hoạt ảnh "nổ" và để tránh người bắn tự giết mình)
private Color color;
// Kiểm tra viên đạn là ảo không. Kẻ địch bắn đạn ảo để kiểm tra xem có đường đi rõ ràng đến người chơi không.
private boolean imaginary;
```

- Các phương thức chính:

  + **draw():** vẽ đạn lên màn hình. Đạn là những hình tròn lồng nhau. Khi mới bắn, khi bắn 1 thời gian, khi đạn nổ sẽ khác nhau. Từ đó tạo hiệu ứng đẹp cho game.

  + **update():** cập nhật vị trí cũng như thời gian sống của đạn. Ví trí được cập nhật bằng cách thay đổi giá trị x,y. Còn thời gian sống của đạn được tăng thêm sau mỗi lần cập nhật.

  + **update(ArrayList<Wall> walls):** Phương thức này xử lý trường hợp đạn di chuyển với tường. Phương thức trả về true nếu đạn còn tồn tại, trả về false nếu đạn bị loại (biến mất).

e) **SeekingBullet:** Tương tự như lớp Bullet, nhưng đạn này có khả năng theo dõi hoặc tìm mục tiêu dựa trên phương thức update(), có thể điều khiển chuyển động và tương tác của viên đạn.

- Một số thuộc tính thêm như

```
final private static float ACCELERATION_RATE = (float)0.1; //gia tốc
final private static float MAX_SPEED = (float)3.0; //tốc độ tối đa

private float aX;//gia tốc theo trục x
private float aY;//gia tốc theo trục y
```

f) **Button:** Lớp này tạo ra các nút bấm trong trò chơi. Nó cung cấp các phương thức để vẽ nút, kiểm tra xem một vị trí củ thể có nằm trong nút hay không, thiết lập vị trí, màu sắc, kích cỡ...

- Các thuộc tính của Button:

```
// các kích thước và màu chuẩn cho một nút trong trò chơi
final public static int STANDARD_WIDTH = 200;
final public static int STANDARD_HEIGHT = 50;
final public static int STANDARD_FONT_SIZE = 20;
final public static Color STANDARD_BOX_COLOR = new Color(40, 120, 10);
final public static Color STANDARD_TEXT_COLOR = Color.WHITE;
// các biến lớp hiển thị số nguyên tương ứng với các vị trí của nút
final public static int CENTER_LOCATION = 0;
final public static int LEFT_LOCATION = 1;
final public static int RIGHT_LOCATION = 2;

private int x, y; // tọa độ của góc trên bên trái của hộp chứa văn bản
final private int width, height, textSize; // width và height là kích thước của hộp (không phải là văn bản)
private String text;
private Color boxColor, textColor;
private int textLocation; // nếu văn bản nên ở bên trái, giữa hoặc bên phải của hộp xung quanh
```

- Các phương thức chính:

+ **drawWithShadow():** vẽ bóng đổ cho nút. Thực chất là vẽ 1 hình chữ nhật màu đen.

+ **draw():** Sử dụng Graphics2D để vẽ khối hộp (box) với màu sắc, kích cỡ, vị trí đặt trước. Đồng thời, in chữ lên trên nút sao cho chữ ở vị trí giữa hộp. Phía dưới là minh họa.

+ **isBounds(int[] Pos):** kiểm tra xem 1 vị trí cụ thể có nằm trong 1 Button hay không. Nếu có trả về True, nếu không thì trả về False.

g) **Vector:** Lớp có vai trò xử lý va chạm, tính toán hướng di chuyển của các đối tượng bằng cách thực hiện các phép toán đối với vector trên không gian hai chiều.

- Các thuộc tính:

```
private float x; //trục x
private float y;//trục y
```

- Các phương thức chính:

```
public static boolean areRotatedRectanglesColliding(Vector[] r1, Vector[] r2){
public static Vector longestParallelVector(Vector[] v){
public static Vector shortestParallelVector(Vector[] v){
public Vector subtract(Vector v){
public Vector multiplyByScalar(float s){
public Vector projectOnto(Vector v){
public float dotProduct(Vector v){
public float getMagnitude(){
```

+ **areRotatedRectanglesColliding():** Phương thức này được sử dụng để kiểm tra xem hai hình chữ nhật có đang va chạm hay không. Phương thức này nhận vào hai mảng Vector mô tả các cạnh của hai hình chữ nhật. Nó sử dụng các phương thức khác như subtract, projectOnto, dotProduct, shortestParallelVector và longestParallelVector để thực hiện kiểm tra va chạm.

+ **longestParallelVector():** Phương thức này tìm và trả về vector dài nhất trong một mảng vector.

+ **shortestParallelVector():** Phương thức này tìm và trả về vector ngắn nhất trong một mảng vector.
+ **subtract():** Phương thức này trừ một vector khác từ vector hiện tại và trả về một vector mới.
+ **multiplyByScalar():** Phương thức này nhân vector với một số hạng để tăng kích thước của vector.
+ **projectOnto():** Phương thức này chiếu vector lên một vector khác và trả về vector kết quả.
+ **dotProduct()**: Phương thức này tính tích vô hướng của hai vector.
+ **getMagnitude():** Phương thức này tính độ lớn của vector, không có hướng.

h) Wall

- Class này dùng để khởi tạo các đối tượng tường(không thể đi xuyên qua) trong trò chơi.Có 2 loại tường trong trò chơi là tường có thể phá và không thể phá.Tường được tạo ra bằng cách đọc các ký tự "-" từ tệp dữ liệu.
- Các thuộc tính

```
private boolean breakable;
private Color color;

//Đây phải là public vì chúng ta đang điền vào hàng phía trên trong Level với màu này
public static Color DEFAULT_COLOR = new Color(100, 100, 100);//màu xám
private static Color BREAKABLE_COLOR = new Color(100, 100, 100, 80);
```

- Các phương thức chính:

  + Hàm khởi tạo:

```
public  Wall(float x, float y) {
    //gọi constructor từ GameObject (khởi tạo vị trí)
    super(x, y);
    breakable = false;
    this.color = DEFAULT_COLOR;
}

// constructor bao gồm một biến boolean xác định xem tường có thể phá hủy hay không
public  Wall(float x, float y, boolean breakable) {
    this(x, y);
    if(breakable){ // nếu đúng là tường có thể phá
        this.breakable = true; // set breakable
        this.color = BREAKABLE_COLOR; // màu xám nhạt là màu có thể phá
    }
}
```

  + **draw()**: vẽ tường lên màn hình với kích thước tương ứng.

  + **isBreakable():** kiểm tra xem tường có thể phá hủy hay không.

i) Score

- Lớp đại diện về điểm số người chơi chơi trong game.
- Thuộc tính: gồm name (String) và score (int).
- Các phương thức chính:

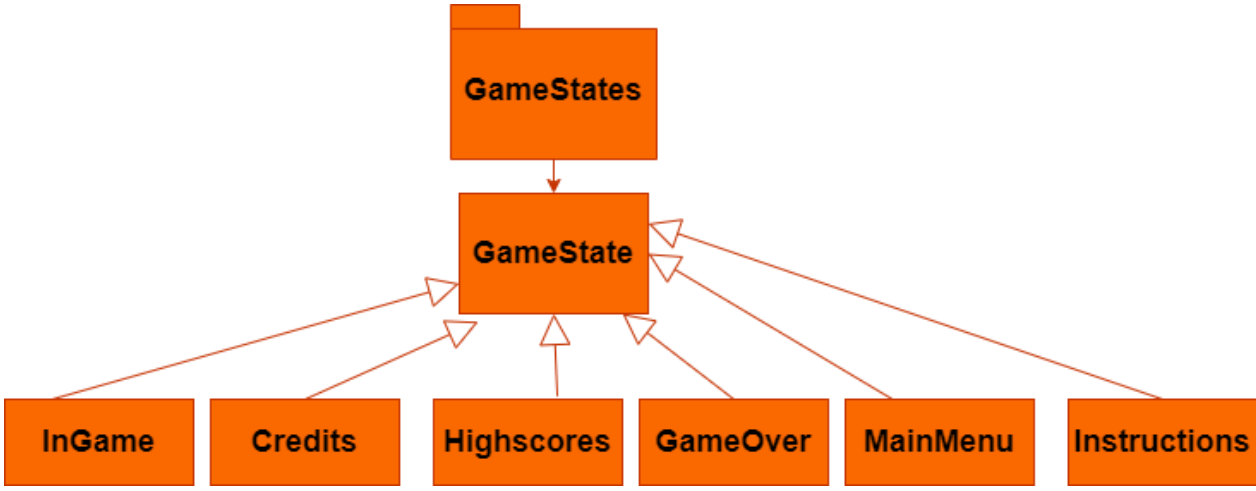
  + Hàm khởi tạo:

```
//Khởi tạo
public Score(){
    name = "";
    score = -1;
}

//Khởi tạo điểm
public Score(String n, int s){
    name = n;
    score = s;
}
```

+ **getScore(), getName(), setScore(), setName():** lấy tên, điểm hoặc thiết lập tên, điểm.

2. **GameStates**

Package này chứa các lớp đại diện cho các trạng thái trong game. Tức là các trang trong game.



a. GameState

- Đây là một abstract class của hầu hết các class trong package GameStates. Lớp sẽ xây dựng các cái chung mà hầu hết các class đều có.
- Các phương thức chính:

+ **drawLego():** Vẽ logo (phần trên cùng) tại các trang game.

+ **getButtons():** trả về ArrayList<Button>.

+ **addButtton:** Thêm button vào ArrayList<Button>.

+ **addReturnToMenuButton:** Thêm button “ReturnToMenu” vì mọi trang đều có nút “ReturnToMenu”.

- Một số phương thức abstract:

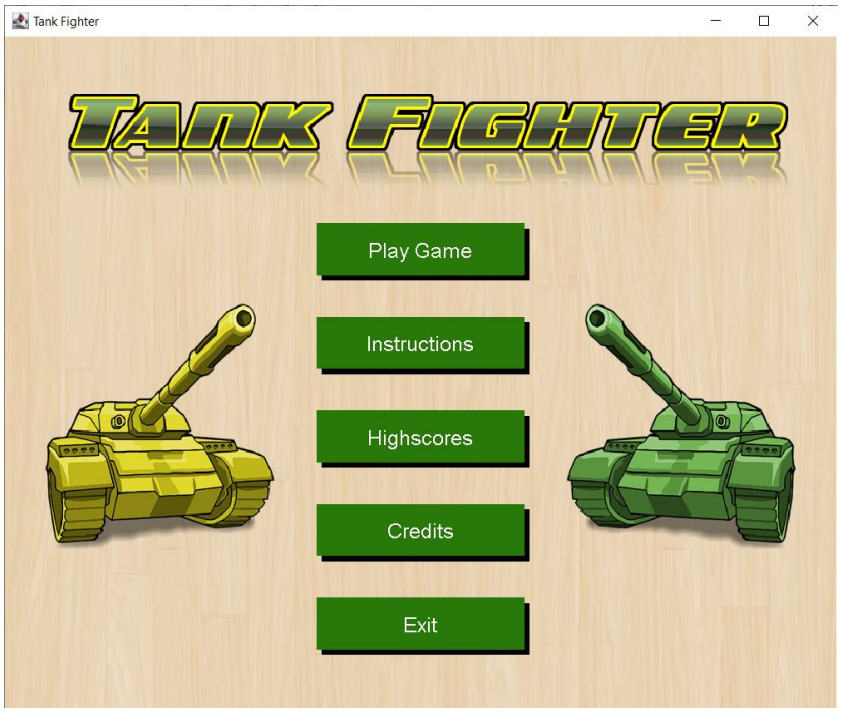
  + **draw():** vẽ đối tượng.

  + **update():** cập nhật trạng thái để hiện thị những cái mới nhất.

  + **resetState():** thiết lập trạng thái về ban đầu.

b. MainMenu

- MainMenu là trang khi bắt đầu vào game. Nó chứa các nút tùy chọn điều hướng như: “Play Game”, “Instructions”, “Highscores”, “Credits”, “Exit”.



- Các phương thức chính:

  + **update():** trả về 1 trạng thái game.

  + **draw():** vẽ titele, logo, hình tank.

```
@Override
public void draw(Graphics2D g, ImageObserver io) {
        //vẽ title
        for (Button b : getButtons()) {
            b.drawWithShadow(g);
        }
        //vẽ logo
        drawLogo(g, io);

        //vẽ hình tank
        BufferedImage greenTank = null;
        BufferedImage yellowTank = null;
        greenTank = ResourceGetter.getBufferedImage(filepath:"greenTankIcon.png");
        yellowTank = ResourceGetter.getBufferedImage(filepath:"yellowTankIcon.png");

        g.drawImage(greenTank, x:520, y:250, width:250, height:250, io);
        g.drawImage(yellowTank, x:30, y:250, width:250, height:250, io);

}
```

+ **resetState():** trả về trạng thái ban đầu.

+ **mousePresed():** Trả về trạng thái game mới khi người dùng nhấn vào các ô lựa chọn.

```
// trả về GameState mới cần truy cập (ví dụ: nếu "Instructions")
public int mousePressed(int[] pos){
    String currButtonText = getMenuButtonPressed(pos);
    if(currButtonText.equals(anObject:"Play Game")){
        return 3;
    }else if(currButtonText.equals(anObject:"Instructions")){
        return 4;
    }else if(currButtonText.equals(anObject:"Highscores")){
        return 2;
    }else if(currButtonText.equals(anObject:"Credits")){
        return 0;
    }else if(currButtonText.equals(anObject:"Exit")){
        return -1;
    }
    //nếu người dùng không nhấn bất kỳ nút nào, hãy tiếp tục ở menu chính
    return 5;
}
```

c. Credits

- Nơi chứa thông tin các thành viên làm ra game này.
- Lớp này chủ yếu chứa nền, logo, ảnh và nút “BackToMenu”.

- Phương thức **draw():**

```
@Override
public void draw(Graphics2D g, ImageObserver io) {
    super.drawLogo(g, io);
    // tải ảnh credit lên
    credit = ResourceGetter.getBufferedImage(filepath:"credits.png");
    g.drawImage(credit, x:0, y:40, observer:null);//tải ảnh credits.png lên màn hình
    getButtons().get(index:0).drawWithShadow(g);// Vẽ button "Back to Main Menu"

}
```

d. Instructions.

- Lớp này chứa ảnh hướng dẫn cách chơi game.
- Lớp này cũng tương tự với lớp Credits nên không phải nói thêm nữa.

e. Highscores.

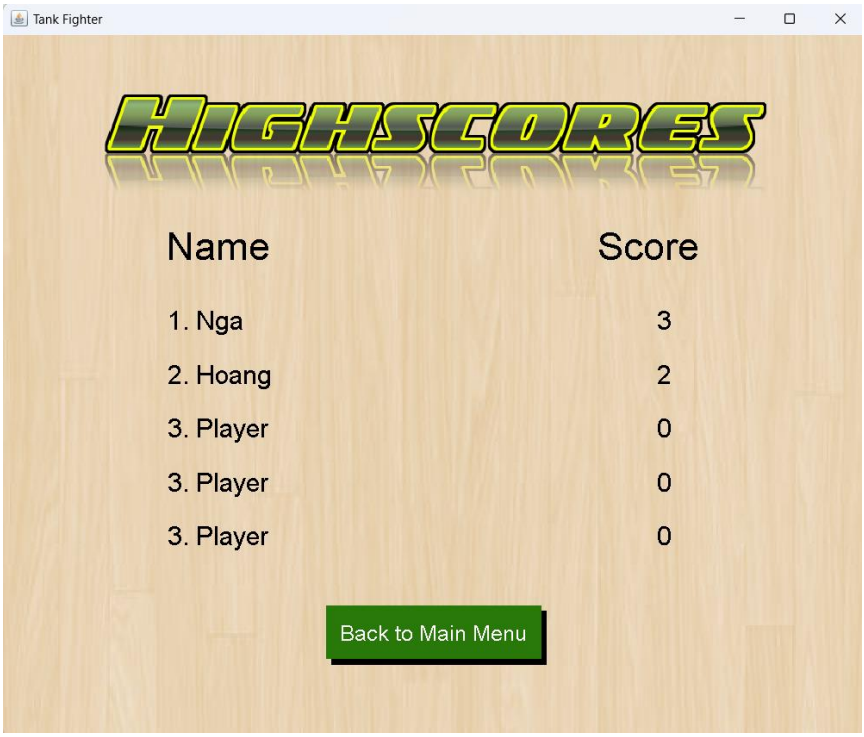
- Lớp này là nơi hiện top 5 điểm cao nhất của người chơi.
- Phương thức chính:

  + **quickSort():** xây dựng thuật toán quicksort để sắp xếp danh sách điểm theo thứ tự giảm dần.

  + **readHighscores():** đọc file “TankFight_Highscores.txt” lưu tên và điểm của các lượt chơi. Sau đó sắp xếp thành 1 danh sách Score theo thứ tự giảm dần sử dụng quicksort().

  + **draw():** hiển thị danh sách top 5 điểm cao nhất.

+ **resetState():** thiết lập trạng thái trang ban đầu. Hình minh họa:



f. InGame

- Là trạng thái chơi game. Lớp này sẽ xử lý xem người chơi qua màn chưa, người chơi đã giành chiến thắng chưa, người chơi khi bị đạn bắn sẽ bị trừ đi 1 máu.
- Các thuộc tính:

```
private int numLevels;
private int currLevelIndex = 0;
private Level currLevel;
//initialLevels lưu trữ trạng thái ban đầu của Cấp độ.
//vì vậy nếu bạn chết ở giữa một cấp độ, Cấp độ đó sẽ được tìm thấy trong initLevels
// để đặt lại trạng thái của Cấp về trạng thái ban đầu
//(ví dụ: thay thế xe tăng ở vị trí ban đầu)
private ArrayList<Level> initialLevels;
private ArrayList<Level> levels;

private int livesLeft;
private boolean beatGame = false;

// những trái tim được vẽ để tượng trưng cho số lần có thể chết
private BufferedImage heartImage = null;
```

- Các phương thức chính:

+ **draw():** vẽ toàn bộ mọi thứ trên màn hình tại level hiện tại.

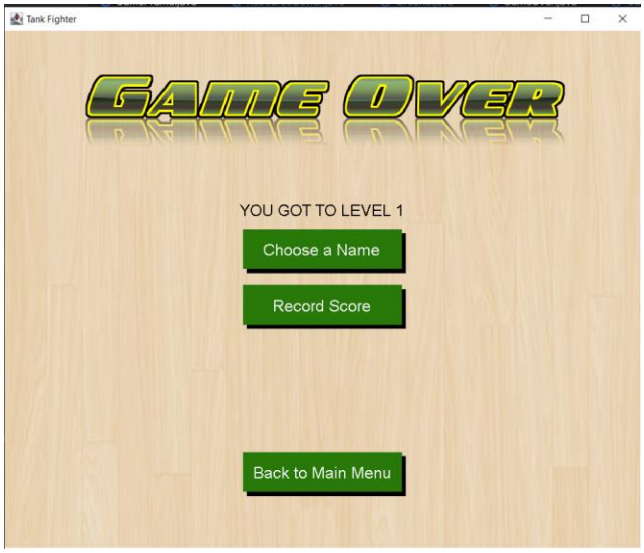
```
public void draw(Graphics2D g, ImageObserver io) {
    currLevel.drawLevel(g, currLevelIndex + 1, io);
    //vẽ tim
    //x vị trí của trái tim chúng ta đang vẽ
    int currHeartX = 290;
    for (int i = 0; i < livesLeft; i++) {
        currHeartX += 50;
        g.drawImage(heartImage, currHeartX, 615, io);

    }
    //vẽ logo
    super.drawLogo(g, io);

}
```

+ **update():** kiểm tra xem người chơi thắng level này chưa để sang level tiếp. Nếu người chơi bị trúng đạn thì mất 1 máu. Nếu máu bằng 0 thì người chơi thua cuộc.

```
public int update() {
    //levelState = 1 nếu người chơi đã vượt qua cấp độ, 2 nếu họ đã chết, 0 nếu ngược lại
    int levelState = currLevel.update();
    String newLogoPath = currLevel.checkChangeLogo();

    //thay đổi logo
    if (newLogoPath != null) {
        if(newLogoPath.equals("levelLogo.png")){
            newLogoPath = "level" + (currLevelIndex + 1) + ".png";
        }
        logoImages.clear();
        logoWidths.clear();
        setLogoImageFromPath(newLogoPath);

    }
    // nếu người chơi vượt qua được cấp độ
    if (levelState == 1) {
        boolean beatGame = nextLevel();
        if(beatGame){
            return GameStateHandler.GAME_OVER_STATE;

        }else{// nếu người chơi vẫn đang chơi
            return GameStateHandler.IN_GAME_STATE;
        }

    }
    //nếu người chơi chết
    else if(levelState == 2){
        // nếu họ vẫn còn mạng thì trừ đi 1 mạng
        if(livesLeft > 1){
            livesLeft--;
            // đặt lại cấp độ
            levels.set(currLevelIndex, initialLevels.get(currLevelIndex).clone());
            currLevel = levels.get(currLevelIndex);
            return 3;
        }
        //nếu hết mạng thì người chơi thua
        else{
```

g. GameOver

- Lớp này đại diện cho trạng thái người chơi thua cuộc. Tại đây người chơi sẽ được thông báo về số cấp độ mà bạn đã vượt qua, chọn tên và lưu lại điểm số của mình. Sau khi điền tên và ấn vào nút “Record Score” người chơi sẽ được đưa đến trạng thái HightScore (nơi hiển thị 5 ngươi chơi có điểm cao nhất). Chú ý, tên và điểm được lưu vào file “TankFight_Highscores.txt”. Hoặc nếu ấn vào nút “Back to Main Menu” thì sẽ được đưa về trạng thái MainMenu như đã giới thiệu ở trên.

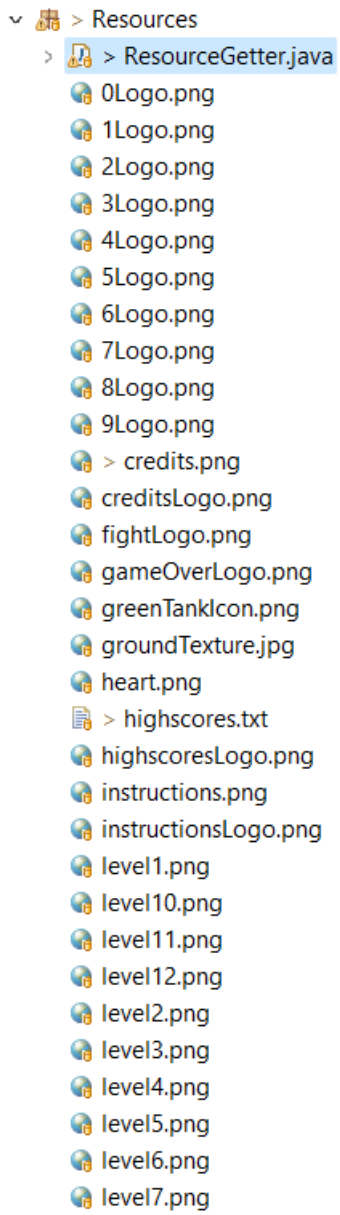


- Phương thức **mousePressed()** được sử dụng khi người dùng nhấn vào các nút.

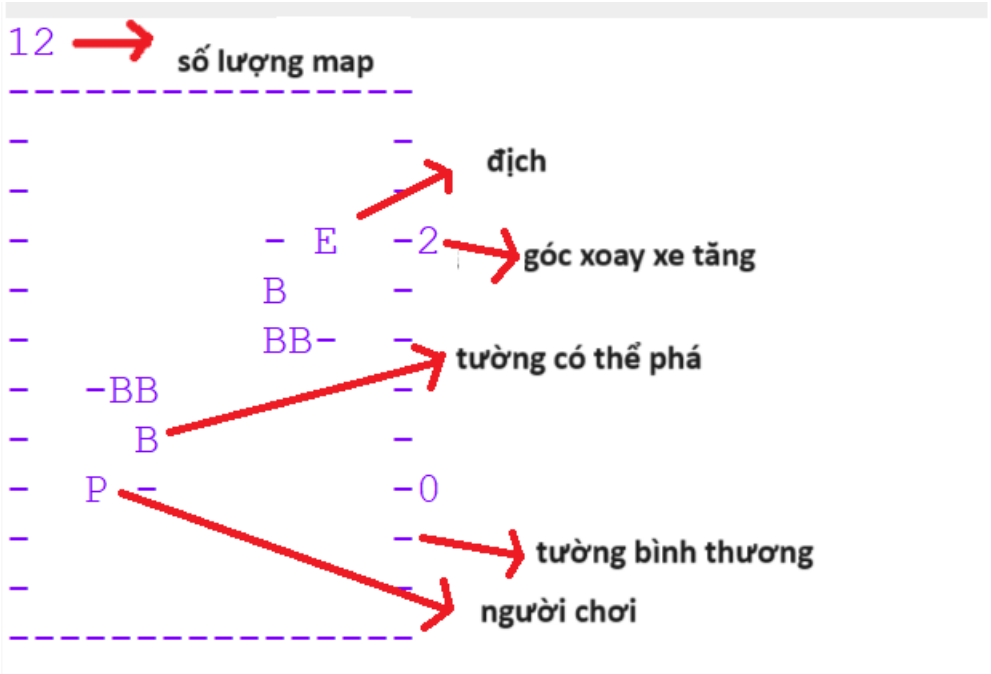
```
public int mousePressed(int[] mousePos, int score, Highscores highscores){
    //nếu người chơi ấn button "return to main menu" thì return 5
    if(getButtons().get(index:3).isInBounds(mousePos)) return 5;
    //Nếu người chơi ấn "enter name here"
    if(getButtons().get(index:1).isInBounds(mousePos)){
        playerName = JOptionPane.showInputDialog(message:"Enter Your Name Below");
        if(playerName == null){
            playerName = "UNNAMED";
        }
        getButtons().get(index:1).setText(playerName);
    }
    //Nếu người chơi ấn "Record Score"
    else if(getButtons().get(index:2).isInBounds(mousePos)){
        //JOptionPane.showMessageDialog(null, "recording score");
        //chúng ta cần kiểm tra xem tên đó có còn rỗng không đề phòng trường hợp họ không nhấn "Choose a Name"
        if(playerName == null){
            playerName = "UNNAMED";
        }
        writeToHighscores(playerName, lastLevel);
        //cho trang điểm cao biết tên và điểm của người chơi hiện tại
        //(để nó có thể được hiển thị và đánh dấu trong trang điểm cao)
        highscores.setRecentPlayerName(playerName);
        highscores.setRecentPlayerScore(score);

        // yêu cầu GameStatesHandler truy cập trang điểm cao
        return 2;

    }
    // nếu người chơi không muốn thay đổi trạng thái trò chơi thì vẫn ở lại màn hình GameOver
    return 1;

}
```

3. Resources

- Đây là package chứa 1 class cung cấp các phương thức đọc file âm thanh, hình ảnh, văn bản... và tài nguyên hình ảnh, âm thanh của game.



- Ví dụ trong file “level.txt” sẽ là toàn bộ thiết kế 12 bản đồ của level dưới dạng ký tự đại diện



- Lớp ResourceGetter: xây dựng các phương thức để đọc file, ghi file. Ví dụ phương thức getHighscoreScanner() trả về 1 đối tượng Scanner để đọc file “Tank_HighScores.txt” chứa tên và điểm của người chơi.

```
public static Scanner getHighscoresScanner(){
    File f2 = new File(ResourceGetter.class.getProtectionDomain().getCodeSource().getLocation().getPath());
    f2 = new File((f2.getParent() + "\\TankFighter_Highscores.txt").replace("%20", " "));
    Scanner s = null;
    if(!f2.exists()){
        try{
            f2.createNewFile();

        }catch(Exception e){
            JOptionPane.showMessageDialog(null, "error creating file");
        }
    }
    //System.out.println(f2);
    try{
        s = new Scanner(f2);
    }catch(FileNotFoundException e){
        JOptionPane.showMessageDialog(null, "error with reading highscores");
    }
    return s;
}
```

4. Tankfighter

a) GameStateHander

- Lớp này quản lý các trạng thái trong game. Ví dụ, các trạng thái game xuất hiện ở phần intro game gồm: Play Game, Instructions, Highscore, Credits, Exit,

GameOver.. Khi chơi thua sẽ hiện trạng thái thua cuộc. còn khi thắng sẽ hiện ảnh victory.

- Các thuộc tính của lớp

```
// Các biến hằng là các kích thước dài,rộng
final public static int FRAME_WIDTH = 800;
final public static int FRAME_HEIGHT = 600;
final public static int FRAME_WIDTH_IN_BLOCKS = 16;
final public static int FRAME_HEIGHT_IN_BLOCKS = 12;
final public static int BLOCK_WIDTH = FRAME_WIDTH / FRAME_WIDTH_IN_BLOCKS;
final public static int BLOCK_HEIGHT = FRAME_HEIGHT / FRAME_HEIGHT_IN_BLOCKS;
// Các số nguyên đại diện cho các trạng thái game
final public static int CREDITS_STATE = 0;
final public static int GAME_OVER_STATE = 1;
final public static int HIGHSCORES_STATE = 2;
final public static int IN_GAME_STATE = 3;
final public static int INSTRUCTIONS_STATE = 4;
final public static int MAIN_MENU_STATE = 5;
```

- Các phương thức:

+ Hàm khởi tạo: khởi tạo 1 danh sách chứa các trạng thái trong game và khởi tạo trạng thái ban đầu là MainMenu.

```
    // Mảng chứa các trạng thái game
    private GameState[] gameStates;
    // trạng thái game hiện tại
    private GameState currGameState;
    // chỉ số của trạng thái game hiện tại
    private int currGameStateIndex;
    // mảng chứa 2 giá trị (x,y) là vị trí con chuột
    private int[] mousePos;// [x, y]

    // Khởi tạo
    public GameStateHandler() {
        gameStates = new GameState[6];
        gameStates[0] = new Credits();
        gameStates[1] = new GameOver();
        gameStates[2] = new Highscores();
        gameStates[3] = new InGame();
        gameStates[4] = new Instructions();
        gameStates[5] = new MainMenu();

        // Tạo trạng thái mặc định là trạng thái MainMenu.
        initGameState(5);

    }
```

+ **update():** dùng để khởi tạo trạng thái tiếp theo

```
// cập nhật trạng thái hiện tại
public void update() {
    int nextState = currGameState.update(); // Trả về chỉ số trạng thái tiếp theo
    if (nextState != currGameStateIndex) {// Nếu trạng thái game bị thay đổi
        // Hiển thị trạng thái mới
        initGameState(nextState);
    }
    // Nếu người chơi thắng thì thay vì hiển thị GameOver thì sẽ hiển thị ảnh Vitory
    if (nextState == GAME_OVER_STATE && ((InGame) gameStates[IN_GAME_STATE]).didBeatGame()) {
        currGameState.setLogoImageFromPath("logos\\victoryLogo.png");
    }
}
```

+**draw():** dùng để hiển thị trạng thái game hiện tại ra màn hình.

```
// vẽ game
public void drawGame(Graphics2D g, ImageObserver io) {
    // vẽ nền
    BufferedImage image = ResourceGetter.getBufferedImage("groundTexture.jpg");

    // Do 50 pixel đầu là phần Level rồi nền vị trí tung độ y phải tăng theo 50
    // pixel
    g.drawImage(image, 0, 0, FRAME_WIDTH, FRAME_HEIGHT + 50, io);

    // vẽ trạng thái game hiện tại
    gameStates[currGameStateIndex].draw(g, io);

}
```

+ **mousePressed():** dùng để xử lý khi người dùng bấm chuột tại các trang. Ví dụ nếu đang ở trang “In Game” thì khi bấm chuột nghĩa là đạn được bắn ra.

b) Level

- Lớp này dùng để tạo mỗi level (màn chơi game). Lớp sẽ chứa các đối tượng chính có trong game như: Player, ArrayList<Enemy>, ArrayList<Wall>,ArrayList<Bullet>.
- Các thuộc tính

```
public class Level {
    //Mỗi 1 level đều gồm có 1 người chơi, các xe địch, tường và đạn.
    private Player player;
    private ArrayList<Enemy> enemies;
    private ArrayList<Wall> walls;
    private ArrayList<Bullet> bullets;

    // map được sử dụng để tìm được đi cho địch. Nó xây dựng vị trí nào mà địch có thể đi được.
    public Node map[][] = new Node[16][12]; // 16 hàng,12 cột

    public long startTime;//Thời gian khi màn game bắt đầu
    public float currentTime;//thời gian kể từ khi bắt đầu chơi màn game

    //Ước lượng thời gian cho 1 khung hình thay đổi. Nó được sử dụng để cập nhật phần Level.
    //Ví dụ nếu thời gian nằm giữa 10 và 4000 thì đó là khoảng thời gian để thay đổi logo
    //từ chữ "Set" sang "Flight"
    private static final int APPROX_TIME_PER_FRAME = 80;
    private static final int INTRO_DELAY = 500;//ms khoảng thời gian giữa các lego

    //Khởi tạo
```

- Các phương thức chính:

  + Hàm khởi tạo:

```
    //Khởi tạo
    public Level(Player player, ArrayList<Enemy> enemies, ArrayList<Wall> walls) {
        this.player = player;
        this.enemies = enemies;
        this.walls = walls;
        startTimer();
        bullets = new ArrayList<>();
        loadMap();
    }
```

  + **update():** trong từng level, sẽ cập nhật danh sách địch, danh sách đạn, đồng thời biết được người chơi thắng hay thua level đó

  + **drawLevel():** Vẽ các đối tượng trong có trong game.

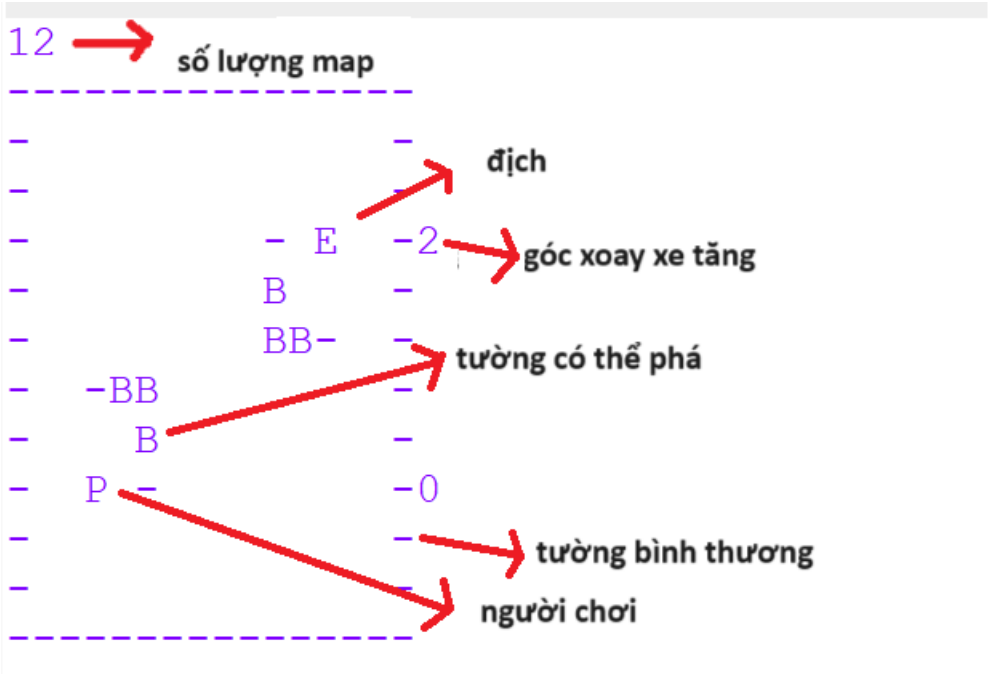
```
    public void drawLevel(Graphics2D g, int levelNumber, ImageObserver io){
        // Tạo 1 hình chữ nhật kích thước 50*800 ở phía trên cùng
        g.fillRect(0, 0, GameStateHandler.FRAME_WIDTH, GameStateHandler.BLOCK_HEIGHT);

        //Vẽ người chơi
        player.draw(g);

        //Vẽ địch
        for (Enemy enemy : enemies) {
            enemy.draw(g);
        }

        //Vẽ từng bức tường
        for (Wall wall : walls) {
            wall.draw(g);
        }

        //Vẽ từng viên đạn
        for (Bullet bullet : bullets) {
            bullet.draw(g);
        }

    }
```

  + **loadMap():** cập nhật đường đi có thể qua của địch. Những vị trí nào có tường thì địch không thể đi qua, còn lại là đều có thể qua được.

c) LevelReader

- Lớp này chịu trách nhiệm khởi tạo từng đối tượng Level bằng việc đọc file “levels.txt”.

- Quy tắc trong file “level.txt”: ví dụ, “E” đại diện cho vị trí địch, “P” đại diện cho vị trí người chơi.



d) Node

- Lớp này chủ yếu được xây dựng để phục vụ cho thuật toán tìm đường đi cho địch sử dụng BFS. Node đại diện cho vị trí của xe tăng ở trên bản đồ.
- Về thiết kế, màn hình chơi game sẽ được chia thành ô lưới, kích cỡ 12*16, kích cỡ mỗi ô là 50*50 pixels.
- Các thuộc tính:

```
private int x, y, cellX, cellY;  //vị trí tọa độ x,y và vị trí ô cellx, celly
private Node parent = null; // node cha
private boolean visited = false;// biến đánh dấu đã thăm hay chưa

public Node(int x, int y) {
    this.x = x;
    this.y = y;
    cellX = (int)Math.round((double)x/50.0); //từ tọa độ đưa về thứ tự ô
    cellY = (int)Math.round((double)y/50.0); //từ tọa độ đưa về thứ tự ô
}
```

e) Pathfinding

- Nguyên lý di chuyển của xe tăng địch là nó luôn di chuyển tới gần người chơi. Vì muốn xe địch đến người chơi nhanh nhất nên nhóm sử dụng giải thuật BFS.

- Lớp này xe xây dựng thuật toán BFS để tìm đường đi ngắn nhất từ địch đến người chơi. Nếu tìm được đường đi thì trả về 1 ArrayList<Node> ls theo chiều từ đích tới nguồn. Ví dụ nếu ls.size()>=2 thì vị trí tiếp theo mà địch cần di chuyển là vị trí ls[ls.size()-2]. Còn nếu ls.size()=1 thì địch ngay bên cạnh người chơi nên không phải di chuyển nữa. Nếu không có đường đi đến địch thì nó sẽ di chuyển đến 1 vị trí ngẫu nhiên nào đó nhưng vẫn áp dụng BFS để đi đến điểm đó.
- Ảnh minh họa thuật toán

```
public void findPath() {
    while (queue.size() > 0) {//lặp đến khi nào hàng đợi rỗng thì thôi

        Node currentNode = queue.get(0);  //Lấy nốt ở đầu hàng đợi
        queue.remove(0);


        if (currentNode.equals(end)) { //nếu gặp được điểm đích
            path.add(currentNode.getParent());  //thêm các điểm cha vào hàng đợi

            while(path.size() > 0 && currentNode.getParent() != null && path.get(path.size()-1).getParent() != null) {
                path.add(path.get(path.size()-1).getParent());
            }
            queue.clear(); //xóa hàng đợi đi để dừng vòng lặp

        }
        //Thêm các nốt xung quanh vào trong hàng đợi.
        checkNeighbours(currentNode);

    }
}
```

f) DrawingSurface

- Kế thừa lớp JPanel để hiển thị giao diện game trên Frame, đồng thời xử lý tương tác thông qua các interface ActionListener, MouseMotionListener, MouseListener.
- Sử dụng 2 đối tượng:

+ GameStateHandler game: Đối tượng chứa giao diện game

+ Timer timer: giúp ta xử lý và thực thi tương tác sau một khoảng thời gian cố định

.

```
public DrawingSurface(){
    game = new GameStateHandler();
    //đọc bàn phím
    addKeyListener(new TAdapter());
    //theo dõi chuyển động của chuột => xác định vị trí của chuột theo tọa độ x, y
    addMouseMotionListener(this);
    //đọc việc nhấn chuột
    addMouseListener(this);
    setFocusable(true);
    setBackground(Color.WHITE);
    timer = new Timer(10, this);//delay per frame (ms). default: 10
    timer.start();
}
```

- Hàm vẽ giao diện

```
private void doDrawing(Graphics g) {
    Graphics2D g2d = (Graphics2D) g;
    game.drawGame(g2d, this);
}
```

- Class xử lý việc nhận tương tác từ bàn phím cho Panel

```
//class xử lý nhập bàn phím
private class TAdapter extends KeyAdapter {

    @Override
    public void keyReleased(KeyEvent e) {
        game.keyReleased(e);
    }

    @Override
    public void keyPressed(KeyEvent e) {
        game.keyPressed(e);
    }
}
```

- Ghi đè phương thức để lấy tọa độ của chuột trên Panel

```
@Override
public void mouseMoved(MouseEvent e) {
    //lấy tọa độ của chuột trên Panel
   int[] pos = {e.getX(), e.getY()};
   game.mouseMoved(pos);
}
```

- **paintComponent()** hiển thị Panel trên Frame

```
@Override
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    doDrawing(g);
}
```

- **actionPerformed()** sau mỗi lần lặp của timer sẽ có tương tác, lúc này ta hiển thị lại panel dựa trên tương tác đó

```
//khi timer hết thời gian thì update lại game và hiển thị lại
@Override
public void actionPerformed(ActionEvent e){
    game.update();
    repaint();
}
```

g) GameFrame

- Lớp này chứa hàm main() để chạy toàn bộ game.

- Lớp kế thừa class Jframe để xây dựng khung.

```
public class GameFrame extends JFrame {
    // Khởi tạo
    public GameFrame() {
        // Tạo ra giao diện game
        initUI();
    }

    // Tạo 1 giao diện game chính
    private void initUI() {
        // Đặt tên game
        setTitle("Tank Fighter");

        // Thêm giao diện chính của game
        add(new DrawingSurface());
        // Thiết lập kích thước cửa sổ hiển thị game
        setSize(816, 689); // Khung bên trong có 800 x 650
        // Khi nhấn nút close thì giao diện game sẽ mất
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Khi giao diện game hiện ra, thì sẽ xuất hiện ở giữa màn hình máy tính
        setLocationRelativeTo(null);
    }
```

- Hàm main(): chạy toàn bộ chương trình

```
public static void main(String[] args) {

    Runnable GameRun = new Runnable() {
        @Override
        public void run() {
            // Tạo ra 1 đối tượng để hiển thị game
            GameFrame windowFrame = new GameFrame();
            // Đảm bảo hiển thị game ra màn hình
            windowFrame.setVisible(true);
        }
    };


    // Tạo một thread mới và bắt đầu nó
    new Thread(GameRun).start();

    Sound gameSound = new Sound("..\\Resources\\tank-battle-13719.wav");

    Runnable soundRun = new Runnable() {
        public void run() {

                gameSound.play();

        }
    };

    // Tạo một thread mới và bắt đầu nó
    new Thread(soundRun).start();

}
```

- Nhóm đã sử dụng **thread** để phân chương trình thành 2 luồng, 1 luồng xử lý game, luồng còn lại xử lý âm thanh.

h) Sound

- Lớp này tạo đối tượng liên quan đến âm thanh
- Thuộc tính:

```
private Clip clip ;// để phát âm thanh
private FloatControl volumeControl;  //điều khiển âm thanh
```

- Phương thức:

+ Hàm khởi tạo:

```
public Sound(String soundFileName) {
    try {
        //Lấy đối tượng âm thanh
        AudioInputStream audioInputStream = AudioSystem.getAudioInputStream(getClass().getResource(soundFileName));
        clip = AudioSystem.getClip();
        clip.open(audioInputStream);
        //Tạo 1 đối tượng kiểu FloatControl để điều chỉnh âm thanh
        volumeControl=(FloatControl) clip.getControl(FloatControl.Type.MASTER_GAIN);
     // Thêm LineListener để theo dõi sự kiện kết thúc âm thanh
        clip.addLineListener(new LineListener() {
            @Override
            public void update(LineEvent event) {
                if (event.getType() == LineEvent.Type.STOP) {
                    clip.setFramePosition(0); // Khi kết thúc, quay lại đầu
                    clip.start();                    }
            }
        });

    } catch (Exception e) {
        System.out.println("Không thể mở file âm thanh: " + soundFileName);
        e.printStackTrace();
    }
}
```

+ **play():** phát nhạc.

```
public void play() {
    if (clip != null) {
        clip.setFramePosition(0);
        decreaseVolume(-30.0f);
        clip.start();
    }
}
```

+ **stop():**dừng nhạc

```
public void stop() {
    if (clip != null) {
        clip.stop();
    }
}
```

+ **decreaseVolume():** giảm âm lượng âm thanh đi .

```
Phương thức giảm âm lượng (volume là giá trị âm lượng mới, ví dụ: -10.0f)
public void decreaseVolume(float volume) {
    if (volumeControl != null) {
        float currentVolume = volumeControl.getValue();
        volumeControl.setValue(currentVolume + volume);
    }
}
```

C. Hoạt động nhóm

Thành viên	Tuấn	Tú	Trí	Hiệp	Hùng
Công việc phân công	Code, báo cáo, thuyết trình	Code, báo cáo	Code, báo cáo, làm slide	Code, báo cáo	Code, báo cáo

D. Tài liệu tham khảo

- Java AWT: https://www.javatpoint.com/java-tutorial
- Chuỗi hướng dẫn lập trình game 2D Java: https://www.youtube.com/playlist?list=PL4rzdwizLaxYmltJQRjq18a9gsSyEQQ-0
- Thuật toán BFS: https://www.geeksforgeeks.org/breadth-first-search-or-bfs-for-a-graph/