TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI
KHOA CÔNG NGHỆ THÔNG TIN
----------o0o----------

BÁO CÁO BÀI TẬP LỚN
CÔNG NGHỆ JAVA

Đề tài: GAME Tank Trouble

Giảng viên hướng dẫn: Vũ Văn Huấn
Nhóm thực hiện: Nhóm 29 - CNTT5 - K62
Thành viên: Lại Xuân Diện
Hồ Văn Minh Nhật

Hà Nội, ngày 2 tháng 5 năm 2023

Lời nói đầu

Công nghệ thông tin ngày càng có vai trò quan trọng trong cuộc sống hàng ngày của chúng ta. Việc ứng dụng CNTT vào các lĩnh vực trong đời sống giúp công việc được tiến

hành nhanh chóng và hiệu quả hơn. Có rất nhiều công việc mới phát triển song song với sự phát triển của CNTT , một trong số đó là hiệu ứng game , hướng đi dịch vụ mang lại hiệu quả kinh tế rất lớn.

Chúng em chọn đề tài " lập trình game Tank Trouble bằng ngôn ngữ java" nhằm tìm hiểu sâu hơn về ngôn ngữ java trong lập trình hướng đối tượng , từ đó viết một ứng dụng cụ thể thực nghiệm làm cơ sở kiến thức và định hướng, kế hoạch xây dựng những ứng dụng game cụ thể , phát triển theo hướng dịch vụ trong tương lai. Phần bài làm của chúng em còn nhiều thiếu sót rất mong nhận được sự góp ý từ thầy cô cùng tất cả mọi người. Chúng em xin chân thành cảm ơn.

I.Giới thiệu đề tài

Với sự phát triển mạnh mẽ của khoa học công nghệ như hiện nay, trò chơi điện tử đang ngày càng trở nên phổ biến trong xã hội, đặc biệt là với lứa tuổi học sinh và thanh thiếu niên. Trò chơi điện tử là những trò chơi giải trí được chơi trên các thiết bị điện tử như tivi, máy tính, điện thoại thông minh... Được thiết kế với hình thức đa dạng và độ hấp dẫn cao với hệ thống đồ họa kích thích thị giác và thao tác mượt mà, trò chơi điện tử tạo hứng thú cho người chơi. Vì vậy, chúng em viết ra game này để giúp người chơi thư giãn và giảm stress sau những giờ học hoặc làm việc mệt mỏi.

- Trong trò chơi này, bạn lái một chiếc xe tăng thông qua một mê cung và bạn phải chiến đấu với một xe tăng địch khác. Nó bao gồm bạn là người đầu tiên bắn và ghi nhớ sự bùng nổ của các viên đạn sẽ nảy ra xung quanh các bức tường. Cố gắng hoàn thành mục tiêu của bạn và không kết thúc tiêu diệt bản thân bạn.

- Cách chơi:

+ Người chơi 1 điều khiển chiếc xe tăng màu xanh, người chơi 2 điều khiển chiếc xe màu đỏ.

+ Người chơi 1 sử dụng các phím A= sang trái , S= lùi , D = sang phải, W= tiến lên , Q = bắn đạn.

+ Người chơi 2 sử dụng các phím tiến lên , lui xuống trên bàn phím và nút Enter= bắn đạn .

II.Cài đặt các lớp class sử dụng trong bài toán:

1.BufferedImageLoader

```
package project;

import java.awt.image.BufferedImage;
import java.io.IOException;

import javax.imageio.ImageIO;

public class BufferedImageLoader {

    private BufferedImage image;
```

```
        public BufferedImage loadImage(String path) throws IOException{
            image = ImageIO.read(getClass().getResource(path));
            return image;
        }
    }
```

2.Bullet

```
package project;
public class Bullet{

    private int timer;
    private int owner;
    private int angle;
    private Point position;
    private static final double bulletSpeed = 4.5;

    private GameEngine engine;

    public Bullet(int player, Point position, int angle, GameEngine e) {
        this.owner = player;
        this.angle = angle;
        this.position = position;
        this.timer = 1000; //Thời gian của đạn là 1000 frame / 60 fps = 16.67 s
        this.engine=e;
    }

    public Point getPosition() {
        return position;
    }
    public int getAngle(){
        return angle;
    }

    public void removeBullet(){//Xóa đạn
        if (this.owner==0){
            engine.player1.decreaseNumberOfBulletsFired();
        }else{
            engine.player2.decreaseNumberOfBulletsFired();
        }
        GameEngine.bulletList.remove(this);
    }
```

```
	public void moveBullet(){
		Point nextPoint = new Point(this.position.getxCoord()
				+ (bulletSpeed * Math.cos(Math.toRadians(90-this.angle))),
				this.position.getyCoord()
				- (bulletSpeed * Math.sin(Math
					.toRadians(90-this.angle))));
		if (wallCrashHorizontal(nextPoint, GameEngine.bulletWidth)){ //Nếu đạn đập vào tường ngang
			flipBulletH();
			nextPoint = new Point(this.position.getxCoord()
				+ (bulletSpeed * Math.cos(Math.toRadians(90-this.angle))),
				this.position.getyCoord());
		}else if (wallCrashVertical(nextPoint, GameEngine.bulletWidth)){ //Nếu đạn đập vào tường dọc
			flipBulletV();
			nextPoint = new Point(this.position.getxCoord(),
				this.position.getyCoord()
				- (bulletSpeed * Math.sin(Math.toRadians(90-this.angle))));
		}else if(cornerCrash(nextPoint,GameEngine.bulletWidth)){
			if(this.currentYSquare()==(int)this.currentYSquare()){
				flipBulletH();
				nextPoint = new Point(this.position.getxCoord()
					+ (bulletSpeed * Math.cos(Math.toRadians(90-this.angle))),
					this.position.getyCoord());
			}else{
				flipBulletV();
				nextPoint = new Point(this.position.getxCoord(),
					this.position.getyCoord()
					- (bulletSpeed * Math.sin(Math.toRadians(90-this.angle))));
			}
		}
		this.position=nextPoint;
	}

	public boolean reduceTimer(){
		timer--;
		if (timer<0){
			this.removeBullet();//Xóa đạn
			return true;
		}
		return false;
```

```
	}

	private void flipBulletV(){
		this.angle=(-this.angle) + 360;
	}

	private void flipBulletH(){
		if (this.angle>180){
			this.angle = -this.angle+540;
		}else{
			this.angle=-this.angle+180;
		}
	}

	public void tankCollision(){
		//Kiểm tra nếu đạn đã va chạm với player1 hoặc player2 chưa
			if (collision(engine.player1)){
				this.removeBullet();
				engine.player1.hit();
			}else if (collision(engine.player2)){
				this.removeBullet();
				engine.player2.hit();
			}

	}

	private boolean collision(Player player){//Kiểm tra va chạm giữa đạn và Player
		//tính khoảng cách từ viên đạn với Player
		double distance = Point.distance(player.getCoordinates(),this.position);
		//trả về nếu có va chạm hay chưa
		return (distance<=(GameEngine.tankWidth/2+GameEngine.bulletWidth/2));
	}

	public double currentXSquare() {
		// Trả về giá trị x của ô vuông mà vị trí Player đang ở
		for (int i =0; i<7;i++){
			if
((GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*i<this.position.getxCoord())&&

	(GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*(i+1))>this.position.getxCoor
d()){
				return i;
			}
```

```
		}
		for (int i =0; i<8;i++){
			if
((GameEngine.wallWidth*(i)+GameEngine.squareWidth*i<=this.position.getxCoord())&&

	(GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*(i+1))>this.position.getxCoor
d()){
				return i-0.5;
			}
		}
		return -1;
	}
	public double currentYSquare() {
		// Trả về giá trị y của ô vuông mà vị trí Player đang ở
		for (int i =0; i<7;i++){
			if
((GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*i<this.position.getyCoord())&&

	(GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*(i+1))>this.position.getyCoor
d()){
				return i;
			}
		}
		for (int i =0; i<8;i++){
			if
((GameEngine.wallWidth*(i)+GameEngine.squareWidth*i<=this.position.getyCoord())&&

	(GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*(i+1))>this.position.getyCoor
d()){
				return i-0.5;
			}
		}
		return -1;
	}

	private boolean wallCrashVertical(Point p, int w){
		//Trả về nếu vật thể có tọa độ p, bán kính w có đang nằm trên tường dọc không
		w=w/2;
		int xWall=0;
		if (this.currentXSquare()!=(int)this.currentXSquare()){
			xWall=1;
		}
		//Kiểm tra có tường bên trái ô vuông không
```

```
            boolean byLeftWall = engine.maze.isWallLeft((int)this.currentXSquare(),
(int)this.currentYSquare());
            //và vật thể đó có nằm trên tường trái không
            boolean inLeftWall = (p.getxCoord()-w<=

        ((int)this.currentXSquare())*GameEngine.squareWidth+((int)this.currentXSquare()+1)*
GameEngine.wallWidth);
            //Tương tự với tường bên phải
            boolean byRightWall = engine.maze.isWallRight((int)this.currentXSquare(),
(int)this.currentYSquare());
            boolean inRightWall = (p.getxCoord()+w>=

        ((int)this.currentXSquare()+1+xWall)*GameEngine.squareWidth+((int)this.currentXSqu
are()+1+xWall)*GameEngine.wallWidth);
            return ((byLeftWall&&inLeftWall)||(byRightWall&&inRightWall));
        }
        private boolean wallCrashHorizontal(Point p, int w){
            //Trả về nếu điểm có tọa độ p, bán kính w có đang nằm trên tường ngang không
            w=w/2;
            int yWall=0;
            if (this.currentYSquare()!=(int)this.currentYSquare()){
                yWall=1;
            }
            //Kiểm tra có tường bên trên ô vuông không
            boolean byTopWall = engine.maze.isWallAbove((int)this.currentXSquare(),
(int)this.currentYSquare());
            boolean inTopWall = (p.getyCoord()-w<=

        ((int)this.currentYSquare())*GameEngine.squareWidth+((int)this.currentYSquare()+1)*
GameEngine.wallWidth);
            //Kiểm tra có tường bên dưới ô vuông không
            boolean byBottomWall = engine.maze.isWallBelow((int) this.currentXSquare(),
(int) this.currentYSquare());
            boolean inBottomWall = (p.getyCoord()+w>=

        ((int)this.currentYSquare()+1+yWall)*GameEngine.squareWidth+((int)this.currentYSqu
are()+1+yWall)*GameEngine.wallWidth);
            return ((byTopWall&&inTopWall)|(byBottomWall&&inBottomWall));
        }
        private boolean cornerCrash(Point p, int w){
            //Trả về nếu vật thể có tọa độ p, bán kính w có đang nằm trên góc gần nhất không
            w=w/2;
            //Tìm góc gần điểm tọa độ p nhất
```

```
		int x=0;
		int y=0;
		int xCounter=(int)p.getxCoord();
		while (xCounter>GameEngine.wallWidth+GameEngine.squareWidth/2){
			xCounter=xCounter-GameEngine.wallWidth-GameEngine.squareWidth;
			x++;
		}
		int yCounter=(int)p.getyCoord();
		while (yCounter>GameEngine.wallWidth+GameEngine.squareWidth/2){
			yCounter=yCounter-GameEngine.wallWidth-GameEngine.squareWidth;
			y++;
		}
		//Tìm xem góc gần nhất đó có tường không, nếu không có trả về false
		boolean
isWallInCorner=engine.maze.isWallAbove(x,y)||engine.maze.isWallLeft(x,y);
		if(x>0){
			isWallInCorner=isWallInCorner||engine.maze.isWallAbove(x-1,y);
		}
		if(y>0){
			isWallInCorner=isWallInCorner||engine.maze.isWallLeft(x,y-1);
		}
		if(!isWallInCorner){
			return false;
		}
		//Tính khoảng cách giữa điểm tọa độ p và góc gần nhất đó
		Point p1=new
Point(x*(GameEngine.wallWidth+GameEngine.squareWidth),y*(GameEngine.wallWidth+Gam
eEngine.squareWidth));
		Point p2=new
Point(x*(GameEngine.wallWidth+GameEngine.squareWidth)+GameEngine.wallWidth,y*(Gam
eEngine.wallWidth+GameEngine.squareWidth));
		Point p3=new
Point(x*(GameEngine.wallWidth+GameEngine.squareWidth),y*(GameEngine.wallWidth+Gam
eEngine.squareWidth)+GameEngine.wallWidth);
		Point p4=new
Point(x*(GameEngine.wallWidth+GameEngine.squareWidth)+GameEngine.wallWidth,y*(Gam
eEngine.wallWidth+GameEngine.squareWidth)+GameEngine.wallWidth);
		double distance1=Point.distance(p1,p);
		double distance2=Point.distance(p2,p);
		double distance3=Point.distance(p3,p);
		double distance4=Point.distance(p4,p);
		double distance=Math.min(distance1,
Math.min(distance2,Math.min(distance3,distance4)));
```

```
                return distance<w;
        }
}
```

3. GameEngine

```
package project;
import java.util.ArrayList;
public class GameEngine {
        public static int player1_score;
        public static int player2_score;
        public static boolean player1_dead = false;
        public static boolean player2_dead = false;
        public static final int tankWidth = 60;//Kích thước xe tăng của Player
        public static final int bulletWidth = 12;//Kích thước đạn
        public static final int squareWidth = 100;//Kích thước ô vuông
        public static final int wallWidth = 10;//Kích thước tường
        public Maze maze;
        public Player player1;
        public Player player2;
        public static ArrayList<Bullet> bulletList = new ArrayList<Bullet>(10);//Tạo mảng chứa
đạn
        public GameEngine() {
                // Khởi tạo
                player1_score = 0;
                player2_score = 0;
                maze = new Maze();
                // Tạo người chơi và spawn ở vị trí ngẫu nhiên
                int x = (int) (Math.random() * 7);
                int y = (int) (Math.random() * 7);
                int x1 = (int) (Math.random() * 7);
                int y1 = (int) (Math.random() * 7);
                while (x1 == x && y1==y) {//nếu vị trí spawn giống nhau thì random lại
                        x1 = (int) (Math.random() * 7);
                        y1 = (int) (Math.random() * 7);
                }

        x=GameEngine.wallWidth+GameEngine.squareWidth/2+(GameEngine.wallWidth+Gam
eEngine.squareWidth)*x;

        y=GameEngine.wallWidth+GameEngine.squareWidth/2+(GameEngine.wallWidth+Gam
eEngine.squareWidth)*y;
```

```
        x1=GameEngine.wallWidth+GameEngine.squareWidth/2+(GameEngine.wallWidth+Ga
meEngine.squareWidth)*x1;

        y1=GameEngine.wallWidth+GameEngine.squareWidth/2+(GameEngine.wallWidth+Ga
meEngine.squareWidth)*y1;
                this.player1 = new Player(0, x, y,this);
                this.player2 = new Player(1, x1, y1,this);
        }
        public void roundOver() {//Khởi tạo game mới

                try {
         Thread.sleep(2000);
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
                if (GameEngine.player1_dead) {
                        GameEngine.player2_score++;
                        GameEngine.player1_dead=false;
                } else if (GameEngine.player2_dead) {
                        GameEngine.player1_score++;
                        GameEngine.player2_dead=false;
                }

                // Tạo người chơi và spawn ở vị trí ngẫu nhiên
                int x = (int) (Math.random() * 7);
                int y = (int) (Math.random() * 7);
                int x1 = (int) (Math.random() * 7);
                int y1 = (int) (Math.random() * 7);
                while (x1 == x && y1==y) {//nếu vị trí spawn giống nhau thì random lại
                        x1 = (int) (Math.random() * 7);
                        y1 = (int) (Math.random() * 7);
                }

        x=GameEngine.wallWidth+GameEngine.squareWidth/2+(GameEngine.wallWidth+Gam
eEngine.squareWidth)*x;

        y=GameEngine.wallWidth+GameEngine.squareWidth/2+(GameEngine.wallWidth+Gam
eEngine.squareWidth)*y;

        x1=GameEngine.wallWidth+GameEngine.squareWidth/2+(GameEngine.wallWidth+Ga
meEngine.squareWidth)*x1;
```

```
        y1=GameEngine.wallWidth+GameEngine.squareWidth/2+(GameEngine.wallWidth+Ga
meEngine.squareWidth)*y1;
                this.player1 = new Player(0, x, y,this);
                this.player2 = new Player(1, x1, y1,this);
                GameEngine.bulletList = new ArrayList<Bullet>(10);
                maze = new Maze();
        }
}
```

4. GameLauncher

```
package project;


import java.awt.Canvas;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.KeyEvent;
import java.awt.geom.AffineTransform;
import java.awt.image.AffineTransformOp;
import java.awt.image.BufferStrategy;
import java.awt.image.BufferedImage;
import java.io.IOException;


import javax.swing.*;
public class GameLauncher extends Canvas implements Runnable {
        private static final long serialVersionUID = 1L;


        private GameEngine engine = new GameEngine();
```

```
	public static final int xDimension=780;
	public static final int yDimension=780;//độ lớn màn hình

	private boolean running=false;
	private Thread thread;

	//Khởi tạo biến ảnh:
	private BufferedImage background;
	private BufferedImage tank1;
	private BufferedImage tank2;
	private BufferedImage bullet;
	private BufferedImage hWall;
	private BufferedImage vWall;

	private boolean[] instructionsArray = new boolean[10];
//W,A,S,D,Q,Lên,Xuống,Trái,Phải,Enter

	private synchronized void start(){//Bắt đầu game
		if (running){
			return;
		}
		running=true;
		thread=new Thread(this);
		thread.start();
	}

	private void init() {//Khởi tạo
```

```
        BufferedImageLoader loader = new BufferedImageLoader();
        try{
                background = loader.loadImage("/background.png");
                tank1 = loader.loadImage("/tank1.png");
                tank2 = loader.loadImage("/tank2.png");
                bullet = loader.loadImage("/bullet.png");
                hWall = loader.loadImage("/hWall.png");
                vWall = loader.loadImage("/vWall.png");
        }catch(IOException e){
                e.printStackTrace();
        }

        addKeyListener(new KeyboardInput(this));

}

private synchronized void stop(){//Dừng game
        if (!running){
                return;
        }

        running=false;
        try {
                thread.join();
        } catch (InterruptedException e) {
                e.printStackTrace();
        }
```

```
        System.exit(1);
}


public void run(){//Chạy game
        init();
        long lastTime = System.nanoTime();
        final double amountOfTicks=60.0;//60FPS
        double ns = 1000000000/amountOfTicks;
        double delta = 0;
        while(running){
                long now = System.nanoTime();
                delta+=(now-lastTime)/ns;
                lastTime=now;
                if(delta>=1){
                        tick();
                        render();
                        delta--;
                }
        }
        stop();
}

private void tick(){//mỗi frame


        //Di chuyển player
```

```
if (instructionsArray[0]){
        engine.player1.goForward();//Tiến
} else if (instructionsArray[2]){
        engine.player1.reverse();//Lùi
}
if (instructionsArray[1]){
        engine.player1.turnLeft();//Xoay trái
}else if (instructionsArray[3]){
        engine.player1.turnRight();//Xoay phải
}
if (instructionsArray[5]){
        engine.player2.goForward();//Tiến
} else if (instructionsArray[7]){
        engine.player2.reverse();//Lùi
}
if (instructionsArray[6]){
        engine.player2.turnLeft();//Xoay trái
}else if (instructionsArray[8]){
        engine.player2.turnRight();//Xoay phải
}
//Hành động bắn
if (instructionsArray[4]){
        engine.player1.shoot();
        instructionsArray[4]=false;//Mỗi lần nhấn nút chỉ bắn 1 lần
}
if (instructionsArray[9]){
        engine.player2.shoot();
```

```
                instructionsArray[9]=false;//Mỗi lần nhấn nút chỉ bắn 1 lần
            }
            //Đạn
            for (int i = 0 ;i<GameEngine.bulletList.size();i++){
                GameEngine.bulletList.get(i).moveBullet();
                boolean removed = GameEngine.bulletList.get(i).reduceTimer();//đếm
ngược timer của đạn
                if (!removed){//Khi đạn chưa hết timer
                    GameEngine.bulletList.get(i).tankCollision();//Kiểm tra va chạm
                }
            }

        }

        private void render(){
            BufferStrategy bs = this.getBufferStrategy();
            if(bs==null){
                createBufferStrategy(3);
                return;
            }
            Graphics g = bs.getDrawGraphics();
            //Vẽ vật thể{

            //Vẽ background:
            g.drawImage(background, 0, 0, this);

            //Vẽ tường
```

```
for (int x = 0; x<7;x++){
    for (int y = 0; y<7; y++){
        if (engine.maze.isWallBelow(x, y)){

g.drawImage(hWall,(GameEngine.squareWidth+GameEngine.wallWidth)*x,(GameEngine.squareWidth+GameEngine.wallWidth)*(y+1),this);
        }
        if (engine.maze.isWallRight(x, y)){

g.drawImage(vWall,(GameEngine.squareWidth+GameEngine.wallWidth)*(x+1),(GameEngine.squareWidth+GameEngine.wallWidth)*y,this);
        }
    }
}
//Vẽ tank
double rotationRequired = Math.toRadians(engine.player1.getDirection());
double locationX = tank1.getWidth() / 2;
double locationY = tank1.getHeight() / 2;
AffineTransform tx = AffineTransform.getRotateInstance(rotationRequired, locationX, locationY);
AffineTransformOp op = new AffineTransformOp(tx, AffineTransformOp.TYPE_BILINEAR);
g.drawImage(op.filter(tank1, null), (int)(engine.player1.getCoordinates().getxCoord()-GameEngine.tankWidth/2), (int)(engine.player1.getCoordinates().getyCoord()-GameEngine.tankWidth/2), this);

AffineTransform tx2 = AffineTransform.getRotateInstance(Math.toRadians(engine.player2.getDirection()), tank2.getWidth() / 2, tank2.getHeight() / 2);
```

```
            AffineTransformOp op2 = new AffineTransformOp(tx2,
AffineTransformOp.TYPE_BILINEAR);
            g.drawImage(op2.filter(tank2, null),
(int)(engine.player2.getCoordinates().getxCoord()-GameEngine.tankWidth/2),
(int)(engine.player2.getCoordinates().getyCoord()-GameEngine.tankWidth/2), this);
            //Vẽ đạn
            for (int i = 0 ;i<GameEngine.bulletList.size();i++){

        g.drawImage(bullet,(int)(GameEngine.bulletList.get(i).getPosition().getxCoord()-
GameEngine.bulletWidth/2),(int)(GameEngine.bulletList.get(i).getPosition().getyCoord()-
GameEngine.bulletWidth/2),this);
            }

            //Hiện vật thể
            g.dispose();
            bs.show();
}
    public void keyPressed(KeyEvent e){
            int key = e.getKeyCode();
            if (key==KeyEvent.VK_W){
                instructionsArray[0]=true;//W = tiến
            }else if (key==KeyEvent.VK_A){
                instructionsArray[1]=true;//A = xoay trái
            }else if (key==KeyEvent.VK_S){
                instructionsArray[2]=true;//S = lùi
            }else if (key==KeyEvent.VK_D){
                instructionsArray[3]=true;//D = xoay phải
            }else if (key==KeyEvent.VK_Q){
                instructionsArray[4]=true;//Q = bắn
```

```
            }else if (key==KeyEvent.VK_UP){
                instructionsArray[5]=true;// Mũi tên lên = tiến
            }else if (key==KeyEvent.VK_LEFT){
                instructionsArray[6]=true;// Mũi tên trái = xoay trái
            }else if (key==KeyEvent.VK_DOWN){
                instructionsArray[7]=true;// Mũi tên xuống = lùi
            }else if (key==KeyEvent.VK_RIGHT){
                instructionsArray[8]=true;// Mũi tên phải = xoay phải
            }else if (key==KeyEvent.VK_ENTER){
                instructionsArray[9]=true;// Enter = bắn
            }else if (key==KeyEvent.VK_SPACE){
                //In ra điểm
                System.out.println("Player 1: "+GameEngine.player1_score + ", Player 2: "+GameEngine.player2_score);
            }
        }
        public void keyReleased(KeyEvent e){//Dừng lại chức năng của các nút khi nhả
            int key = e.getKeyCode();
            if (key==KeyEvent.VK_W){
                instructionsArray[0]=false;
            }else if (key==KeyEvent.VK_A){
                instructionsArray[1]=false;
            }else if (key==KeyEvent.VK_S){
                instructionsArray[2]=false;
            }else if (key==KeyEvent.VK_D){
                instructionsArray[3]=false;
            }else if (key==KeyEvent.VK_Q){
```

```
                instructionsArray[4]=false;
            }else if (key==KeyEvent.VK_UP){
                instructionsArray[5]=false;
            }else if (key==KeyEvent.VK_LEFT){
                instructionsArray[6]=false;
            }else if (key==KeyEvent.VK_DOWN){
                instructionsArray[7]=false;
            }else if (key==KeyEvent.VK_RIGHT){
                instructionsArray[8]=false;
            }else if (key==KeyEvent.VK_ENTER){
                instructionsArray[9]=false;
            }
        }


        public static void main(String args[]){
            GameLauncher game = new GameLauncher();


            //Kích thước cửa sổ
            game.setPreferredSize(new
Dimension(GameLauncher.xDimension,GameLauncher.yDimension));
            game.setMaximumSize(new
Dimension(GameLauncher.xDimension,GameLauncher.yDimension));
            game.setMinimumSize(new
Dimension(GameLauncher.xDimension,GameLauncher.yDimension));
            JFrame frame = new JFrame("Tank");
            frame.add(game);
            frame.pack();
```

```
                frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                frame.setResizable(false);
                frame.setLocationRelativeTo(null);
                frame.setVisible(true);
                game.start();
        }
}
```

5. KeyBoardInput

```
package project;

import java.awt.event.KeyEvent;
import java.awt.event.KeyAdapter;

public class KeyboardInput extends KeyAdapter {
        private GameLauncher game;
        public KeyboardInput(GameLauncher game){
                this.game=game;
        }
        public void keyPressed(KeyEvent e){
                game.keyPressed(e);
        }
        public void keyReleased(KeyEvent e){
                game.keyReleased(e);
        }
}
```

6. Maze

```
package project;
```

```
import java.util.ArrayList;

public class Maze {

private static final int gridWidth = 7;

//Số ô vuông mỗi cạnh = 7

private static boolean[][][] walls = new boolean[2][gridWidth + 1][gridWidth + 1];

//mảng trả về true nếu có tường, trả về false nếu không có tường

//walls[0=ngang, 1=dọc][x][y]

//Ô vuông vị trí (a,b) sẽ có:

//tường trái = [1][a][b]

//tường phải = [1][a+1][b]

//tường trên = [0][b][a]

//tường dưới = [0][b+1][a]

public Maze() {

new Maze(30);

//số tường = 30

}

public Maze(int density) {

walls = generateMaze(density);

//tạo một bảng tường ngẫu nhiên

Square[][] areas = this.mergeAll();

while (areas.length>1){//tìm tất cả các khu vực cách biệt nhau

removeWall(areas[0]);//xóa bỏ tường giữa 2 ô vuông cạnh nhau
```

```
areas = this.mergeAll();
}
}
public boolean isWallLeft(int x, int y) {
//Kiểm tra xem ô vuông có tường trái không, có thì trả về true, không có trả về false
return walls[1][x][y];
}
public boolean isWallRight(int x, int y) {
//Kiểm tra xem ô vuông có tường phải không, có thì trả về true, không có trả về falsee
return walls[1][x + 1][y];
}
public boolean isWallAbove(int x, int y) {
//Kiểm tra xem ô vuông có tường trên không, có thì trả về true, không có trả về false
return walls[0][y][x];
}
public boolean isWallBelow(int x, int y) {
//Kiểm tra xem ô vuông có tường dưới không, có thì trả về true, không có trả về false
return walls[0][y + 1][x];
}
private Square[] neighbors(Square square) {
// trả về tất cả các ô vuông có cùng canh với ô vuông đang kiểm tra và bao gồm cả chính nó
Square[] neighbors = new Square[5];
```

```
int numberOfNeighbors = 0;
if (!isWallLeft(square.getxCoord(), square.getyCoord())) {
neighbors[numberOfNeighbors] = new Square(square.getxCoord() - 1,
square.getyCoord());
numberOfNeighbors++;
}
if (!isWallRight(square.getxCoord(), square.getyCoord())) {
neighbors[numberOfNeighbors] = new Square(square.getxCoord() + 1,
square.getyCoord());
numberOfNeighbors++;
}
if (!isWallAbove(square.getxCoord(), square.getyCoord())) {
neighbors[numberOfNeighbors] = new Square(square.getxCoord(),
square.getyCoord() - 1);
numberOfNeighbors++;
}
if (!isWallBelow(square.getxCoord(), square.getyCoord())) {
neighbors[numberOfNeighbors] = new Square(square.getxCoord(),
square.getyCoord() + 1);
numberOfNeighbors++;
}
neighbors[numberOfNeighbors] = square;
```

```
// Xóa bỏ giá trị null
int realLength = 0;
boolean found = false;
for (int i = 0; i < neighbors.length; i++) {
if (neighbors[i] == null) {
realLength = i;
found = true;
break;
}
}
if (found) {
Square[] cutReturnValue = new Square[realLength];
for (int i = 0; i < realLength; i++) {
cutReturnValue[i] = neighbors[i];
}
return cutReturnValue;
} else {
return neighbors;
}
}
private static boolean isNeighbour(Square a, Square b){
//Kiểm tra xem 2 ô vuông có là hàng xóm của nhau không
```

```
int x = a.getxCoord();

int y = a.getyCoord();

int x2 = b.getxCoord();

int y2 = b.getyCoord();

return (x2-1==x && y2==y)||(x2+1==x && y2==y)||(x2==x && y2-1==y)||(x2==x &&
y2+1==y);

}

private void removeWall(Square[] area){

// Kết nối một khu vực với khu vực khác

Square[] neighbors = areaNeighbors(area);

//Tìm tất cả các hàng xóm của khu vực đó

int numberToRemove=2;//Xóa bỏ 1 đến 2 bức tường

for (int j = 0; j<numberToRemove;j++){

int randomNum = (int) (Math.random()*neighbors.length);

Square chosen = neighbors[randomNum];

//Lựa chọn ô vuông ngẫu nhiên trong danh sách hàng xóm

Square areaSquareChosen = new Square(0,0);

for (int i = 0; i<area.length; i++){

if (isNeighbour(area[i],chosen)){

areaSquareChosen=area[i];

break;

}

}
```

```
//Tìm ô vuông trong khu vực kề với ô vuông đã chọn
int x = chosen.getxCoord();
int y = chosen.getyCoord();
int x2 = areaSquareChosen.getxCoord();
int y2 = areaSquareChosen.getyCoord();
if (x==x2){
walls[0][Math.max(y,y2)][x]=false;
}else{
walls[1][Math.max(x2, x)][y]=false;
}
}
//Xóa bỏ tường giữa 2 ô vuông đó
}
private static Square[] merge(Square[] a, Square[] b) {
Square[] returnValue = new Square[b.length + a.length];
for (int i = 0; i < a.length; i++) {
if (!(a[i] == null)) {
returnValue[i] = a[i];
}
}// Trả về các giá trị trong a khác null
for (int i = 0; i < b.length; i++) {// Kiểm tra tất cả giá tri trong b
boolean found = false;
```

```
for (int j = 0; j < a.length; j++) {
if (!(a[j] == null) && !(b[i] == null) && b[i].equals(a[j])) {
found = true;
}
}// Kiểm tra liệu giá trị đó cũng có trong a không
if (!found && !(b[i] == null)) {// Nếu không thì trả về giá trị đó
for (int j = 0; j < returnValue.length; j++) {
if (returnValue[j] == null) {
returnValue[j] = b[i];
break;
}
}
}
}
// Xóa bỏ giá trị null
int realLength = 0;
boolean found = false;
for (int i = 0; i < returnValue.length; i++) {
if (returnValue[i] == null) {
realLength = i;
found = true;
break;
```

```
}

}

if (found) {

Square[] cutReturnValue = new Square[realLength];

for (int i = 0; i < realLength; i++) {

cutReturnValue[i] = returnValue[i];

}

return cutReturnValue;

} else {

return returnValue;

}

}

private static Square[] intersection(Square[] a, Square[] b) {

Square[] returnValue = new Square[b.length];

for (int i = 0; i < b.length; i++) {//Kiểm tra tất cả giá tri trong b

boolean found = false;

for (int j = 0; j < a.length; j++) {

if (!(a[j] == null) && !(b[i] == null) && b[i].equals(a[j])) {

found = true;

}

}// Kiểm tra liêu giá tri đó cũng có trong a không

if (found && !(b[i] == null)) {// Nếu có thì trả về giá tri đó
```

```
for (int j = 0; j < returnValue.length; j++) {
if (returnValue[j] == null) {
returnValue[j] = b[i];
break;
}
}
}
}
// Xóa bỏ giá trị null
int realLength = 0;
boolean found = false;
for (int i = 0; i < returnValue.length; i++) {
if (returnValue[i] == null) {
realLength = i;
found = true;
break;
}
}
if (found) {
Square[] cutReturnValue = new Square[realLength];
for (int i = 0; i < realLength; i++) {
cutReturnValue[i] = returnValue[i];
```

```
}
return cutReturnValue;
} else {
return returnValue;
}
}
private Square[] areaNeighbors(Square[] area){
//Trả về tất cả hàng xóm của khu vực đang được chọn
Square[] neighbors = new Square[49];
//Trả về tất cả hàng xóm có chung tường
for (int i = 0; i<area.length;i++){
neighbors = merge(neighbors,walledNeighbours(area[i]));
}
//Xóa ô vuông đã có trong khu vực
Square[] cutNeighbors = new Square[neighbors.length];
int cutNeighborsLength=0;
for (int i = 0;i<neighbors.length;i++){
boolean found = false;
for (int j=0;j<area.length;j++){
if (neighbors[i].equals(area[j])){
found = true;
}
```

```
}
if (!found){
cutNeighbors[cutNeighborsLength]=neighbors[i];
cutNeighborsLength++;
}
}
//Xóa bỏ giá trị null
Square[] returnValue = new Square[cutNeighborsLength];
for (int i = 0; i<cutNeighborsLength;i++){
returnValue[i]=cutNeighbors[i];
}
return returnValue;
}
private Square[] walledNeighbours(Square square){
//Trả về tất cả ô vuông kề cạnh và có chung tường với ô vuông được chọn
Square[] neighbors = new Square[4];
int numberOfNeighbors = 0;
if (isWallLeft(square.getxCoord(), square.getyCoord())&&square.getxCoord()-1>=0) {
neighbors[numberOfNeighbors] = new Square(square.getxCoord() - 1,
square.getyCoord());
numberOfNeighbors++;
}
```

```
if (isWallRight(square.getxCoord(), square.getyCoord())&&square.getxCoord()+1<7) {
neighbors[numberOfNeighbors] = new Square(square.getxCoord() + 1,
square.getyCoord());
numberOfNeighbors++;
}
if (isWallAbove(square.getxCoord(), square.getyCoord())&&square.getyCoord()-1>=0) {
neighbors[numberOfNeighbors] = new Square(square.getxCoord(),
square.getyCoord() - 1);
numberOfNeighbors++;
}
if (isWallBelow(square.getxCoord(), square.getyCoord())&&square.getyCoord()+1<7) {
neighbors[numberOfNeighbors] = new Square(square.getxCoord(),
square.getyCoord() + 1);
numberOfNeighbors++;
}
// Xóa bỏ giá tri null
int realLength = 0;
boolean found = false;
for (int i = 0; i < neighbors.length; i++) {
if (neighbors[i] == null) {
realLength = i;
found = true;
```

```
break;
}
}
if (found) {
Square[] cutReturnValue = new Square[realLength];
for (int i = 0; i < realLength; i++) {
cutReturnValue[i] = neighbors[i];
}
return cutReturnValue;
} else {
return neighbors;
}
}
private Square[][] connections() {
// Trả về danh sách các ô vuông mà mỗi ô vuông kết nối
Square[][] connections = new Square[49][5];
for (int i = 0; i < 7; i++) {
for (int j = 0; j < 7; j++) { // Tìm hàng xóm của mỗi ô vuông
connections[i + 7 * j] = neighbors(new Square(i, j));
}
}
return connections;
```

```
}

private Square[][] mergeAll() {//Trả về danh sách các khu vực tách biệt

Square[][] set = connections();//Cho danh sách các ô vuông mà mỗi ô vuông kết nối vào một mảng rồi chuyển sang List

ArrayList<Square[]> list = new ArrayList<Square[]>(49);

for (int i = 0; i < 49; i++) {

list.add(set[i]);

}

//Hợp các khu vực có chung ô vuông

for (int i = 0; i<list.size()-1;i++){

for (int j = (i+1); j<list.size();j++){

if (intersection(list.get(i), list.get(j)).length != 0) {

//Nếu kết nối có chung ô vuông thì hợp chúng thành một

list.set(i, merge(list.get(i), list.get(j)));

list.remove(j);

i=0;

j=0;

}

}

}

//chuyển tất cả về dạng mảng

Square[][] returnValue = new Square[list.size()][49];

for (int i = 0; i < list.size(); i++) {
```

```
returnValue[i]=list.get(i);
}
return returnValue;
}
private boolean[][][] generateEmptyMaze() { //Tạo mê cung rỗng
boolean[][][] walls = new boolean[2][gridWidth + 1][gridWidth + 1];
for (int i = 0; i < 7; i++) {
walls[0][0][i] = true;
walls[0][7][i] = true;
walls[1][0][i] = true;
walls[1][7][i] = true;
}
return walls;
}
private boolean[][][] generateMaze(int density) { //Thêm tường vào mê cung
boolean[][][] walls2 = generateEmptyMaze();
for (int j = 0; j < 2; j++) {
for (int i = 0; i < density; i++) {
int randomNumber = (int) (Math.random() * (7));
int randomNumber2 = (int) (Math.random() * (7));
walls2[j][randomNumber][randomNumber2] = true;
}
```

```
}
return walls2;
}
public String toString() { //In ra mê cung
String[] lines = new String[8];
lines[0] = " ";
for (int i = 0; i < 7; i++) {
lines[0] = lines[0] + (walls[0][0][i] ? "_" : " ") + " ";
}
for (int i = 1; i < 8; i++) {
lines[i] = "";
for (int j = 0; j < 8; j++) {
lines[i] = lines[i] + (walls[1][j][i - 1] ? "|" : " ")
+ (walls[0][i][j] ? "_" : " ");
}
}
String toReturn = "";
for (int i = 0; i < 8; i++) {
toReturn = toReturn + lines[i] + "\n";
}
return toReturn;
}
```

```
public static void main(String[] args) {
Maze maze = new Maze(30);
System.out.print(maze);
}
}
```

7. Player

```
 package project;
//import java.util.Map;
public class Player {
private int playerNumber; // Player1 hoặc Player2
private int numberOfBulletsFired; // Lưu số đạn đã bắn của xe tăng của Player
private int direction; // hướng mà Player đang nhìn
// 0 là góc 12h, 180 là góc 6h
private Point coordinates; // Tọa độ điểm trung tâm của xe tăng của Player
private static final int rotationSpeed = 3; // Tốc độ xoay xe tăng của Player
private static final double forwardSpeed = 2; // Tốc độ tiến xe tăng của Player
private static final double reverseSpeed = -1.3; // Tốc độ lùi xe tăng của Player
private GameEngine engine;
public void turnRight() {// Xoay sang phải
this.direction += rotationSpeed;
if (this.direction>359){
this.direction-=360;
```

```
}

}

public void turnLeft() {// Xoay sang trái

this.direction -= rotationSpeed;

if (this.direction<0){

this.direction+=360;

}

}

private void move(double speed) {

Point nextPoint = new Point(this.coordinates.getxCoord()

+ (speed * Math.cos(Math.toRadians(90-this.direction))),

this.coordinates.getyCoord()

- (speed * Math.sin(Math

.toRadians(90-this.direction))));

if (cornerCrash(nextPoint,GameEngine.tankWidth)){//Nếu Player di chuyển vào góc

nextPoint = new Point(this.coordinates.getxCoord()

+ (speed * Math.cos(Math.toRadians(90-this.direction))),

this.coordinates.getyCoord());//Đẩy theo hướng ngang

if(cornerCrash(nextPoint,GameEngine.tankWidth)||wallCrashVertical(nextPoint,GameEngine.tankWidth)||wallCrashHorizontal(nextPoint,GameEngine.tankWidth)){

nextPoint = new Point(this.coordinates.getxCoord(),//Nếu không thể đẩy sang ngang thì đẩy theo hướng dọc

this.coordinates.getyCoord()
```

```
- (speed * Math.sin(Math.toRadians(90-this.direction))));

if(cornerCrash(nextPoint,GameEngine.tankWidth)||wallCrashVertical(nextPoint,GameEngine.tankWidth)||wallCrashHorizontal(nextPoint,GameEngine.tankWidth)){

nextPoint= new Point(this.coordinates.getxCoord(),//Nếu không đẩy được thì đứng yên

this.coordinates.getyCoord());

}

}

}else {

if
(wallCrashVertical(nextPoint,GameEngine.tankWidth)&&wallCrashHorizontal(nextPoint,GameEngine.tankWidth)){

nextPoint= new Point(this.coordinates.getxCoord(),//Nếu không đẩy được thì đứng yên

this.coordinates.getyCoord());

}

if (wallCrashVertical(nextPoint, GameEngine.tankWidth)){ //Nếu Player đi vào tường dọc

nextPoint = new Point(this.coordinates.getxCoord(),//Đẩy Player theo hướng doc

this.coordinates.getyCoord()

- (speed * Math.sin(Math.toRadians(90-this.direction))));

}

if(wallCrashHorizontal(nextPoint,GameEngine.tankWidth)){//Nếu Player đi vào tường ngang

nextPoint = new Point(this.coordinates.getxCoord()//Đẩy Player theo hướng ngang

+ (speed * Math.cos(Math.toRadians(90-this.direction))),

this.coordinates.getyCoord());

}
```

```
}

this.coordinates=nextPoint;

}

public void goForward(){//Tiến lên

move(forwardSpeed);

}

public void reverse() {//Lùi lại

move(reverseSpeed);

}

private boolean wallCrashVertical(Point p, int w){

//Trả về nếu vật thể có tọa độ p, bán kính w có đang nằm trên tường dọc không

w=w/2;

int xWall=0;

if (this.currentXSquare()!=(int)this.currentXSquare()){

xWall=1;

}

//Kiểm tra có tường bên trái ô vuông không

boolean byLeftWall = engine.maze.isWallLeft((int)this.currentXSquare(),
(int)this.currentYSquare());

//và vật thể đó có nằm trên tường trái không

boolean inLeftWall = (p.getxCoord()-w<=

((int)this.currentXSquare())*GameEngine.squareWidth+((int)this.currentXSquare()+1)*Game
Engine.wallWidth);
```

```
//Tương tự với bên phải

boolean byRightWall = engine.maze.isWallRight((int)this.currentXSquare(),
(int)this.currentYSquare());

boolean inRightWall = (p.getxCoord()+w>=

((int)this.currentXSquare()+1+xWall)*GameEngine.squareWidth+((int)this.currentXSquare()+
1+xWall)*GameEngine.wallWidth);

return ((byLeftWall&&inLeftWall)||(byRightWall&&inRightWall));

}

private boolean wallCrashHorizontal(Point p, int w){

//Trả về nếu vật thể có toạ độ p, bán kính w có đang nằm trên tường ngang không

w=w/2;

int yWall=0;

if (this.currentYSquare()!=(int)this.currentYSquare()){

yWall=1;

}

//Kiểm tra có tường bên trên ô vuông không

boolean byTopWall = engine.maze.isWallAbove((int)this.currentXSquare(),
(int)this.currentYSquare());

boolean inTopWall = (p.getyCoord()-w<=

((int)this.currentYSquare())*GameEngine.squareWidth+((int)this.currentYSquare()+1)*Game
Engine.wallWidth);

//Kiểm tra có tường bên dưới ô vuông không

boolean byBottomWall = engine.maze.isWallBelow((int) this.currentXSquare(), (int)
this.currentYSquare());

boolean inBottomWall = (p.getyCoord()+w>=
```

```
((int)this.currentYSquare()+1+yWall)*GameEngine.squareWidth+((int)this.currentYSquare()+
1+yWall)*GameEngine.wallWidth);

return ((byTopWall&&inTopWall)|(byBottomWall&&inBottomWall));

}

private boolean cornerCrash(Point p, int w){

//Trả về nếu vật thể có tọa độ p, bán kính w có đang nằm trên góc gần nhất không

w=w/2;

//Tìm góc gần điểm tọa độ p nhất

int x=0;

int y=0;

int xCounter=(int)p.getxCoord();

while (xCounter>GameEngine.wallWidth+GameEngine.squareWidth/2){

xCounter=xCounter-GameEngine.wallWidth-GameEngine.squareWidth;

x++;

}

int yCounter=(int)p.getyCoord();

while (yCounter>GameEngine.wallWidth+GameEngine.squareWidth/2){

yCounter=yCounter-GameEngine.wallWidth-GameEngine.squareWidth;

y++;

}

//Tìm xem góc gần nhất đó có tường không, nếu không có trả về false

boolean isWallInCorner=engine.maze.isWallAbove(x,y)||engine.maze.isWallLeft(x,y);

if(x>0){
```

```
isWallInCorner=isWallInCorner||engine.maze.isWallAbove(x-1,y);

}

if(y>0){

isWallInCorner=isWallInCorner||engine.maze.isWallLeft(x,y-1);

}

if(!isWallInCorner){

return false;

}

//Tính khoảng cách giữa điểm tọa độ p và góc gần nhất đó

Point p1=new
Point(x*(GameEngine.wallWidth+GameEngine.squareWidth),y*(GameEngine.wallWidth+Ga
meEngine.squareWidth));

Point p2=new
Point(x*(GameEngine.wallWidth+GameEngine.squareWidth)+GameEngine.wallWidth,y*(Ga
meEngine.wallWidth+GameEngine.squareWidth));

Point p3=new
Point(x*(GameEngine.wallWidth+GameEngine.squareWidth),y*(GameEngine.wallWidth+Ga
meEngine.squareWidth)+GameEngine.wallWidth);

Point p4=new
Point(x*(GameEngine.wallWidth+GameEngine.squareWidth)+GameEngine.wallWidth,y*(Ga
meEngine.wallWidth+GameEngine.squareWidth)+GameEngine.wallWidth);

double distance1=Point.distance(p1,p);

double distance2=Point.distance(p2,p);

double distance3=Point.distance(p3,p);

double distance4=Point.distance(p4,p);

double distance=Math.min(distance1, Math.min(distance2,Math.min(distance3,distance4)));
```

```
return distance<w;

}

// Phương thức bắn đạn

public void shoot() {

if (this.numberOfBulletsFired < 5) {

//Nơi đạn xuất phát

Point bulletStart = new Point(this.coordinates.getxCoord()

+ ((GameEngine.bulletWidth/2+GameEngine.tankWidth/2) * Math.cos(Math.toRadians(90-
this.direction))),

this.coordinates.getyCoord()

- ((GameEngine.bulletWidth/2+GameEngine.tankWidth/2) * Math.sin(Math.toRadians(90-
this.direction))));

if (wallCrashVertical(bulletStart,
GameEngine.bulletWidth)||wallCrashHorizontal(bulletStart,GameEngine.bulletWidth)||cornerCr
ash(bulletStart,GameEngine.bulletWidth)){ //if the bullet would start in/over any of the walls

this.hit();

} else {

GameEngine.bulletList.add(new Bullet(playerNumber, bulletStart, direction,engine));

//Tao thực thể đạn mới, với playerNumber là Player bắn, di chuyển theo direction hiên tai và toa
độ xuất phát bulletStart

this.numberOfBulletsFired += 1;//Tăng số đạn đã bắn lên

}

}

}

// Phương thức trúng đạn
```

```
public void hit() {

if (this.playerNumber == 0) {

GameEngine.player1_dead=true;

} else {

GameEngine.player2_dead=true;

}

engine.roundOver();

}

public double currentXSquare() {

// Trả về giá trị x của ô vuông mà vị trí Player đang ở

for (int i =0; i<7;i++){

if
((GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*i<this.coordinates.getxCoord())&
&

(GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*(i+1))>this.coordinates.getxCoord(
)){

return i;

}

}

for (int i =0; i<8;i++){

if
((GameEngine.wallWidth*(i)+GameEngine.squareWidth*i<=this.coordinates.getxCoord())&&

(GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*(i+1))>this.coordinates.getxCoord(
)){

return i-0.5;
```

```
}

}

return -1;

}

public double currentYSquare() {

// Trả về giá trị y của ô vuông mà vị trí Player đang ở

for (int i =0; i<7;i++){

if
((GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*i<this.coordinates.getyCoord())&
&

(GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*(i+1))>this.coordinates.getyCoord(
)){

return i;

}

}

for (int i =0; i<8;i++){

if ((GameEngine.wallWidth*(i)+GameEngine.squareWidth*i<this.coordinates.getyCoord())&&

(GameEngine.wallWidth*(i+1)+GameEngine.squareWidth*(i+1))>this.coordinates.getyCoord(
)){

return i-0.5;

}

}

return -1;

}
```

```
// Hàm tạo
public Player(int playerNo, double x, double y,GameEngine e) {
this.playerNumber = playerNo;
this.numberOfBulletsFired = 0;
this.direction = (int) (Math.random() * 360);//Player quay về hướng ngẫu nhiên
this.coordinates = new Point(x, y);
this.engine=e;
}
// Getter và Setter
public void decreaseNumberOfBulletsFired() {
while (this.numberOfBulletsFired!=0){
this.numberOfBulletsFired -= 1;
}
}
public Point getCoordinates(){
return this.coordinates;
}
public int getDirection(){
return this.direction;
}
public void setCoordinates(double x, double y){
this.coordinates=(new Point(x,y));
```

```
}
public int getBulletsFired(){
return this.numberOfBulletsFired;
}
}
```

8. Point

```
package project;
public class Point {
private double xCoord;
private double yCoord;//Toa độ điểm là (xCoord, yCoord)
//Hàm tạo
public Point(double a, double b){
this.setxCoord(a);
this.setyCoord(b);
}
//Getter và Setter
public double getxCoord() {
return xCoord;
}
public void setxCoord(double xCoord) {
this.xCoord = xCoord;
}
```

```
public double getyCoord() {
return yCoord;
}
public void setyCoord(double yCoord) {
this.yCoord = yCoord;
}
public void setCoords(double xCoord, double yCoord){
this.xCoord=xCoord;
this.yCoord=yCoord;
}
public String toString(){//Trả về tọa độ
return "" + xCoord + "," + yCoord;
}
public boolean equals(Point a){
return this.getxCoord()==a.getxCoord()&&this.getyCoord()==a.getyCoord();
}
public static double distance(Point p1, Point p2){//Khoảng cách 2 điểm
double xDistance = p1.getxCoord()-p2.getxCoord();
double yDistance = p1.getyCoord()-p2.getyCoord();
return Math.sqrt(xDistance*xDistance+yDistance*yDistance);
}
}
```

9. Square

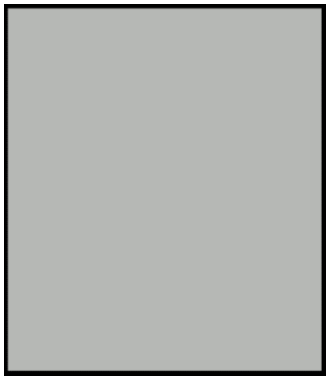
```
package project;

public class Square {

private int xCoord;

private int yCoord;//Toa đô ô vuông là (xCoord, yCoord)

//Hàm tạo

public Square(int a, int b){

this.xCoord=a;

this.yCoord=b;

}

//Getter và Setter

public int getxCoord() {

return xCoord;

}

public int getyCoord() {

return yCoord;

}

public String toString(){

return "" + xCoord + "," + yCoord;

}

public boolean equals(Square a){

return this.getxCoord()==a.getxCoord()&&this.getyCoord()==a.getyCoord();
```

}

}

10. Refereced Libraries

- background.png



- Bullet.png

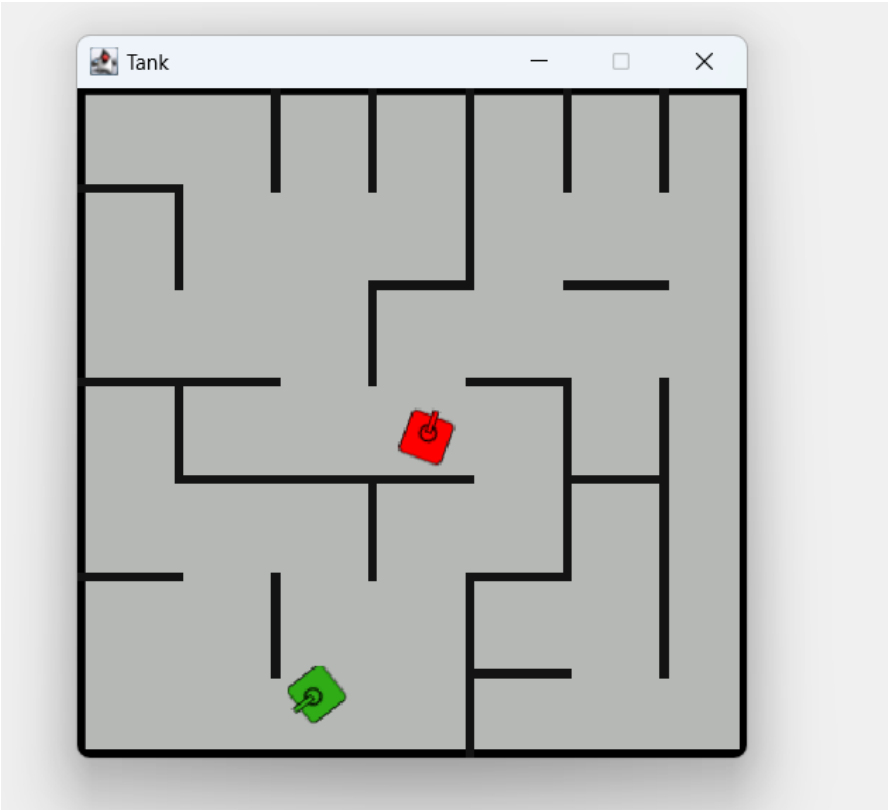
-hWall.png

-tank1.png

-tank2.png

-vWall.png

III. Giao diện chơi game



--0o--HẾT--o0--