TRƯỜNG ĐẠI HỌC ĐIỆN LỰC

**KHOA CÔNG NGHỆ THÔNG TIN**

Logo, company name

Description automatically generated

**BÁO CÁO CHUYÊN ĐỀ HỌC PHẦN**

**LẬP TRÌNH JAVA**

**ĐỀ TÀI:**

**LẬP TRÌNH GAME ĐUA XE 2D**

|  |  |
| --- | --- |
|  |  |
| |  |  | | --- | --- | | **Sinh viên thực hiện** | **: NGUYỄN CHÍ CƯỜNG**  **TRẦN TIẾN HƯNG**  **ĐỖ VĂN PHONG** | | **Giảng viên hướng dẫn** | **: TRẦN HỮU PHI** | | | **Ngành** | **: CÔNG NGHỆ THÔNG TIN** | | | **Chuyên ngành** | **: CÔNG NGHỆ PHẦN MỀM** | | | **Lớp** | **: D16CNPM4** | | | **Khóa** | **: 2021-2026** | | |  |

**Hà Nội, tháng .. năm ....**

1. **Mô tả bài toán và phương hướng giải quyết bài toán**

* Lập trình phần mềm trò chơi đua xe với các yêu cầu:
  + Sử dụng kỹ thuật lập trình hướng đối tượng
  + Có các chướng ngại vật trên đường đi
  + Khi người chơi đạt đến mức điểm nhất định sẽ tự động nâng độ khó của trò chơi

1. **Mô tả bài toán**

* Game đua xe trên máy tính cần phải có:
  + Khởi tạo trò chơi: Là một “Frame” cho phép người chơi biết một số thông tin cần thiết.
  + Người điều khiển xe: Người chơi sử dụng máy tính để diều khiển xe.
  + Đường đua: Nơi người chơi tham gia. Trên đường đua gồm có người chơi và các chướng ngại vật mà người chơi cần phải vượt qua.
  + Vỉa hè: Là những rào cản bên ngoài, nơi có nhiệm vụ giới hạn xe của người chơi đi trên vị trí lòng đường quy định, không được phép vượt ra khỏi phạm vi này.
  + Chướng ngại vật: Một điều không thể thiếu khi nói đến các trò chơi đua xe. Nếu xe tham gia đường đua gặp phải chướng ngại vật trên đường gây ra va chạm thì xe của người chơi sẽ không thể tiếp tụ đi được nữa.

1. **Phương hướng giải quyết bài toán**

* Trên cơ sở đã phân tích của bài toán, dựa trên kiểu lập trình hướng đối tượng, chúng em bắt tay vào xây dựng những lớp, những thuộc tính, những phương thức tương ứng với khi phân tích bài toán.
  + Khởi tạo trò chơi: Cho phép người chơi khởi tạo trò chơi mới.
  + Đường đua và vỉa hè: Để tạo đường đua và vỉa hè trong chương trình đã load những file có tên “road.png”,”background.png”, … từ thư mục chứa các file “.png” của máy.
  + Xe ô tô: Trong chương trình này, phương tiện tham gia chỉ có duy nhất là xe ô tô, nhưng ô tô này là những file ảnh được load từ từ thư mục “Img”. Ảnh khác nhau sẽ tương ứng với ứng với xe khác nhau, xe riêng của người chơi là duy nhất, những chướng ngại vật cũng là những file ảnh khác biệt để dễ phân biệt chứng với xe của người chơi.

1. **Các tính năng của ứng dụng**
2. **Khởi tạo trò chơi.**

Lớp **RaceGame** là lớp dùng để khởi tạo chò trơi. Lớp này có giao

diện đồ họa dễ dàng sử dụng. Đầu tiên, phương thức constructFrame() gọi các

phương thức để đặt kích thước cửa sổ, đặt vị trí, đặt thuộc tính đóng cửa sổ khi

đóng ứng dụng, và bật hiển thị của cửa sổ.

Sau đó, phương thức tạo các đối tượng hình ảnh đường đua, ô tô, và các

chướng ngại vật (obstacles) được tải từ thư mục chứa các file "png" của

máy. Đối tượng "points" là một trường văn bản để hiển thị điểm số của người

chơi

Tiếp theo, phương thức này đặt vị trí và kích thước của ô tô và các chướng

ngại vật, thêm chúng vào đối tượng "racetrack" để hiển thị.

Sau đó, đối tượng "racetrack" được thêm vào nội dung của cửa sổ và gói lại

nội dung để cập nhật lại kích thước của cửa sổ.

Cuối cùng, phương thức bật hiển thị của cửa sổ.

Để có được giao diện và thực hiện những hành động đó lớp RaceGame

đã kế thừa từ lớp JFrame và lắng nghe sự kiện ActionListener.

private void constructFrame(){

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

setResizable(false);

setLocation(350,0);

addKeyListener(this);

racetrack = new JLabel(new ImageIcon(RaceGame.class.getResource("images/race\_track.jpg")));

car = new JLabel(new ImageIcon(RaceGame.class.getResource("images/car.jpg")));

obstacle\_one = new JLabel(new ImageIcon(RaceGame.class.getResource("images/obstacle\_one.png")));

obstacle\_two = new JLabel(new ImageIcon(RaceGame.class.getResource("images/obstacle\_two.png")));

obstacle\_three = new JLabel(new ImageIcon(RaceGame.class.getResource("images/obstacle\_three.png")));

points = new JTextField();

car.setBounds(x, y, 100, 200);

points.setBounds(480, 12, 100, 30);

racetrack.add(car);

racetrack.add(points);

racetrack.add(obstacle\_one);

racetrack.add(obstacle\_two);

racetrack.add(obstacle\_three);

getContentPane().add(racetrack);

pack();

setVisible(true);

}

Phương thức **startRace()** được sử dụng để bắt đầu trò chơi đua xe. Đầu tiên, phương thức tạo một danh sách các đối tượng Obstacle bằng cách khởi tạo ba đối tượng Obstacle với ba hình ảnh được cung cấp và thêm chúng vào danh sách.

Sau đó, nó tạo một đối tượng ThreadControl với danh sách Obstacle, car và points là các tham số và đối tượng này được sử dụng để kiểm soát chuyển động của các chướng ngại vật và điểm số.

Cuối cùng, vòng lặp vô hạn được sử dụng để giữ cho trò chơi chạy, mỗi vòng lặp sẽ tạm dừng trong 70ms.

public void startRace(){

ArrayList<Obstacle> obstacles = new ArrayList();

Obstacle o1 = new Obstacle(obstacle\_one);

Obstacle o2 = new Obstacle(obstacle\_two);

Obstacle o3 = new Obstacle(obstacle\_three);

obstacles.add(o1); obstacles.add(o2); obstacles.add(o3);

new ThreadControl(obstacles, car, points);

while(true){

try {

Thread.sleep(70);

} catch (InterruptedException ex) {

Logger.getLogger(RaceGame.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

1. **Bắt đầu trò chơi.**

Lớp **Car** một lớp được định nghĩa để tạo ra các đối tượng ô tô trong trò chơi đua xe. Lớp này kế thừa từ lớp Thread và chứa một số thuộc tính như sau:

car: là một đối tượng JLabel để hiển thị hình ảnh của ô tô.

rand: là một đối tượng của lớp Random để sinh số ngẫu nhiên.

position\_x: là vị trí x hiện tại của ô tô trên trục x.

position\_y: là vị trí y hiện tại của ô tô trên trục y.

**2.1 Hiển thị ô tô:**

Phương thức **run()** được sử dụng để chạy ô tô trên màn hình. Trong phương thức run(), ô tô được di chuyển từ trên xuống dưới màn hình, bắt đầu từ một vị trí ngẫu nhiên trên trục x và một vị trí cố định trên trục y. Sau khi di chuyển đến đáy màn hình, ô tô sẽ được đặt lại ở một vị trí mới trên trục x và bắt đầu di chuyển lại từ đầu. Thời gian di chuyển của ô tô được tính toán dựa trên vị trí hiện tại của nó trên trục x.

public class Car extends Thread{

JLabel car;

Random rand = new Random();

int position\_x;

int position\_y;

public Car(JLabel car, int position\_x){

this.car = car;

this.position\_x = position\_x;

start();

}

public void run(){

while(true) {

position\_x = rand.nextInt(250) + 150;

position\_y = -100;

try{

sleep(position\_x \* 5 + 1000);

}catch (InterruptedException ex){

Logger.getLogger(Car.class.getName()).log(Level.SEVERE, null, ex);

}

car.setBounds(position\_x, position\_y, 50, 50);

while (car.getY() < 600){

car.setBounds(position\_x, position\_y, 50, 50);

position\_y += 3;

try{

sleep(position\_x / 20);

}catch (InterruptedException ex){

Logger.getLogger(Car.class.getName()).log(Level.SEVERE,null, ex);

}

}

}

}

Lớp **Obstacle** kế thừa từ lớp **Thread**, chứa các thuộc tính và phương thức để tạo và di chuyển vật cản trong game.

obstacle: là một đối tượng JLabel, đại diện cho chướng ngại vật.

rand: là một đối tượng Random, được sử dụng để tạo ra các giá trị ngẫu nhiên.

position\_x: là vị trí ngang của chướng ngại vật.

position\_y: là vị trí dọc của chướng ngại vật.

Phương thức khởi tạo Obstacle nhận đối số là một JLabel đại diện cho vật cản. Khi khởi tạo, lớp Obstacle sẽ tự động bắt đầu chạy một luồng.

**2.2 Di chuyển vật cản**

Phương thức **run()** là phương thức chính của lớp Obstacle, được thực hiện khi luồng bắt đầu chạy. Trong vòng lặp vô hạn, đầu tiên, lớp sẽ tạo ngẫu nhiên một vị trí ban đầu cho vật cản tại position\_x và position\_y. Sau đó, lớp sử dụng sleep() để chờ một khoảng thời gian ngẫu nhiên trước khi bắt đầu di chuyển vật cản xuống dưới.

Trong vòng lặp tiếp theo, vật cản được di chuyển xuống dưới màn hình theo một tốc độ ngẫu nhiên, được tính bằng cách chia một số ngẫu nhiên randNumOne cho một số ngẫu nhiên khác randNumTwo. Khi vật cản đến đáy màn hình, nó sẽ bắt đầu lại từ đầu, tạo một vị trí ban đầu ngẫu nhiên mới và di chuyển xuống dưới màn hình một lần nữa.

public class Obstacle extends Thread{

JLabel obstacle;

Random rand = new Random();

public int position\_x;

public int position\_y;

public Obstacle(JLabel obstacle){

this.obstacle = obstacle;

start();

}

public void run(){

while(true) {

position\_x = rand.nextInt(200) + 150;

position\_y = -300;

double randNumOne = Math.random() \* 300 + 200;

double randNumTwo = Math.random() \* 40 + 10;

try{

sleep(position\_x ^3 );

}catch (InterruptedException ex){

Logger.getLogger(Car.class.getName()).log(Level.SEVERE, null, ex);

}

obstacle.setBounds(position\_x, position\_y, 100, 200);

while (obstacle.getY() < 700){

obstacle.setBounds(position\_x, position\_y, 100, 200);

position\_y += 3;

try{

sleep((long) (randNumOne / randNumTwo));

}catch (InterruptedException ex){

Logger.getLogger(Car.class.getName()).log(Level.SEVERE,null, ex);

}

}

}

}

Lớp **Racing** kế thừa lớp **JFrame** và cài đặt giao diện **KeyListener** để lắng nghe các sự kiện phím bấm từ người chơi. Các thành phần giao diện của trò chơi bao gồm:

JLabel racetrack: hiển thị hình ảnh đường đua

static JLabel car: hiển thị hình ảnh chiếc xe đang chơi

static JLabel obstacle\_one: hiển thị hình ảnh chướng ngại vật trên đường đua

JTextField points: hiển thị số điểm người chơi đã ghi được

Hàm khởi tạo của lớp **Racing** kế thừa từ lớp **JFrame** và đặt tiêu đề cho cửa sổ là "CarRaceGame". Sau đó, nó gọi hàm **constructFrame()** để tạo nội dung cho cửa sổ.

**2.3 Hiển thị giao diện trò chơi**

Phương thức **constructFrame()** thiết lập kích thước và thuộc tính đóng cửa sổ cho frame và thêm một key listener. Sau đó, tạo ra các JLabel để hiển thị các ảnh cho đường đua, xe hơi và chướng ngại vật. Đặt vị trí của xe hơi và chướng ngại vật và thêm chúng vào label của đường đua. Đồng thời, hiển thị điểm số của người chơi. Sau đó, nó thêm hình ảnh của đường đua vào content pane của frame, thêm hình ảnh của chướng ngại vật vào hình ảnh của đường đua, tạo đóng gói cho frame và hiển thị nó lên màn hình.

**2.4 Di chuyển vật cản và cập nhật điểm số**

Phương thức **obstacleMove()** được sử dụng để di chuyển vật cản và cập nhật điểm số của người chơi. Phương thức này bao gồm các chi tiết cụ thể sau:

Thay đổi nội dung của JTextField points thành chuỗi "Score: " cộng với giá trị hiện tại của biến score, và tăng biến score lên một đơn vị.

Di chuyển vật cản xuống dưới bằng cách cập nhật tọa độ Y của nó.

Kiểm tra nếu vật cản đã đi qua đích, đặt lại tọa độ ban đầu cho vật cản tại một vị trí ngẫu nhiên nằm trong khoảng xác định với tọa độ X, và đặt lại tọa độ Y để bắt đầu lại vòng lặp.

public void obstacleMove() {

points.setText("Score: " + String.valueOf(score++));

obstacle\_one.setBounds(obstacle\_one.getX(), obstacle\_one.getY()+10,100,200);

if (obstacle\_one.getY() > 900){

int obstacle\_one\_X = (int) (Math.random()\*200+200);

obstacle\_one.setBounds(obstacle\_one\_X, -50, 100 ,200);

}

}

**2.5 Kiểm tra va chạm**

Phương thức **judgeStop()** để kiểm tra xem xe có va chạm với chướng ngại vật hay không. Bằng cách khởi tạo biến flag là false để đánh dấu rằng chưa có va chạm xảy ra. Sau đó, phương thức sử dụng các điều kiện để xác định xem vị trí của xe có nằm trong vùng giới hạn của chướng ngại vật hay không. Nếu có va chạm xảy ra, phương thức sẽ gọi phương thức interrupted() để dừng trò chơi thực thi hiện tại và gán giá trị flag là true.

public boolean judgeStop(){

boolean flag = false;

boolean carLeftBoundary = car.getBounds().getX() >= obstacle\_one.getX() - 50;

boolean carRightBoundary = car.getBounds().getX() <= obstacle\_one.getX() + 50;

boolean carUpBoundary = car.getBounds().getY() >= obstacle\_one.getY() - 100;

boolean carBelowBoundary = car.getBounds().getY() <= obstacle\_one.getY() + 100;

if(carLeftBoundary && carRightBoundary && carUpBoundary && carBelowBoundary){

Thread.interrupted();

flag = true;

}

Return flag;

}

Lớp ThreadControl kế thừa từ lớp Thread và có các thuộc tính gồm danh sách các chướng ngại vật (obstacles), một đối tượng JLabel cho xe, một đối tượng JTextField để hiển thị số điểm, một vị trí y ban đầu cho các chướng ngại vật và số lượng chướng ngại vật hiện tại (num).

Hàm khởi tạo một đối tượng ThreadControl khi được gọi và thêm các thông tin ô tô, chướng ngại vật và điểm số vào đối tượng ThreadControl.

**2.6 Thông báo kết thúc**

Phương thức có tên **failureNotice**() được gọi khi xe va chạm với chướng ngại vật, dừng tất cả các chướng ngại vật và làm cho xe không thể di chuyển được đồng thời hiển thị thông báo khi game kết thúc .

Trong phương thức này, trước tiên, vòng lặp for được sử dụng để lặp lại tất cả các đối tượng Obstacle trong danh sách obstacles, đồng thời gọi phương thức stop() để dừng chúng.

Sau đó, một đối tượng JFrame được tạo ra để hiển thị hộp thoại thông báo kết thúc game. Trong hộp thoại thông báo, có hai nút tùy chọn được tạo ra: "Exit Game" và "Cancel". Khi người dùng nhấp vào nút "Exit Game", hệ thống sẽ thoát khỏi game và đóng tất cả các cửa sổ. Khi người dùng nhấp vào nút "Cancel", hộp thoại thông báo sẽ được đóng lại. Nội dung của thông báo bao gồm một câu thông báo "Game Over! Your score:" và số điểm đạt được.

public void failureNotice(){

for(Obstacle obstacle: obstacles){

obstacle.stop();

car.disable();

}

JFrame frame = new JFrame();

JButton b1 = new JButton("Exit Game");

JButton b2 = new JButton("Cancel");

b1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e){

System.exit(0);

}});

b2.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

frame.setVisible(false);

}

});

Object[] options = {b1, b2};

JOptionPane.showOptionDialog(frame,"Game Over! Your score: " + points.getText() , "CarRaceGame",

JOptionPane.YES\_NO\_CANCEL\_OPTION, JOptionPane.QUESTION\_MESSAGE,null, options, options[0]);

}

}

**2.7 Kiểm tra di chuyển và Tính điểm**

Phương thức **run()** được gọi khi đối tượng của lớp **ThreadControl** được khởi tạo và bắt đầu chạy. Nó có một vòng lặp vô hạn để theo dõi việc di chuyển của các vật cản và tính điểm cho người chơi. Trong vòng lặp, vị trí của xe được so sánh với vị trí của mỗi vật cản để kiểm tra va chạm. Nếu xe chạm vào bất kỳ vật cản nào, phương thức **failureNotice()** sẽ được gọi để hiển thị thông báo kết thúc trò chơi và dừng tất cả các vật cản. Khi không có va chạm, vòng lặp sẽ tạm dừng trước khi lặp lại.

public void run(){

while(true){

points.setText("Score: " + String.valueOf(num++));

for(Obstacle o: obstacles){

position\_y = o.position\_y;

boolean carLeftBoundary = car.getBounds().getX() >= o.position\_x - 50;

boolean carRightBoundary = car.getBounds().getX() <= o.position\_x + 50;

boolean carUpBoundary = car.getBounds().getY() > o.position\_y - 150;

boolean carBelowBoundary = car.getBounds().getY() < o.position\_y + 100;

if(carLeftBoundary && carRightBoundary && carUpBoundary && carBelowBoundary ){

failureNotice();

return;

}

}

try {

sleep(100);

} catch (InterruptedException ex) {

Logger.getLogger(ThreadControl.class.getName()).log(Level.SEVERE, null, ex);

}

}

}

**2.8 Xử lý sự kiện từ bàn phím**

Phương thức keyPressed(KeyEvent e) ghi đè trong Java để xử lý sự kiện khi một phím được nhấn. Nó kiểm tra xem phím nào đã được nhấn và di chuyển một hình ảnh ô tô trên màn hình dựa trên phím được nhấn và giới hạn về chiều rộng và chiều cao của ô tô trên màn hình. Nếu phím mũi tên bên phải được nhấn và vị trí x hiện tại của ô tô chưa vượt quá giới hạn bên phải (360), ô tô sẽ di chuyển sang phải. Tương tự, nếu phím mũi tên bên trái được nhấn và vị trí x hiện tại của ô tô chưa vượt quá giới hạn bên trái (135), ô tô sẽ di chuyển sang trái . Nếu phím mũi tên lên được nhấn và vị trí y hiện tại của ô tô chưa vượt quá giới hạn phía trên (0), ô tô sẽ di chuyển lên . Nếu phím mũi tên xuống được nhấn và vị trí y hiện tại của ô tô chưa vượt quá giới hạn phía dưới (700), ô tô sẽ di chuyển xuống .

public void keyPressed(KeyEvent e) {

if(e.getKeyCode() == VK\_RIGHT && x < 360) {car.setBounds(x += 10, y, 100, 100);}

if(e.getKeyCode() == VK\_LEFT && x > 135) {car.setBounds(x -= 10, y, 100, 100);}

if(e.getKeyCode() == VK\_UP && y > 0) {car.setBounds(x, y -= 10, 100, 100);}

if(e.getKeyCode() == VK\_DOWN && y < 700) {car.setBounds(x, y += 10, 100, 100);

}

}

1. **Mô tả API**
2. **Thư viện java.util.Random**

Thư viện java.util.Random cung cấp các phương thức để tạo ra các giá trị ngẫu nhiên. Các phương thức này cho phép lập trình viên tạo ra các giá trị ngẫu nhiên với một phân phối đồng nhất hoặc phân phối theo Gauss.

Một ưu điểm của thư viện này là các giá trị ngẫu nhiên tạo ra đều không được lặp lại trong chu kỳ ngẫu nhiên. Điều này giúp đảm bảo tính ngẫu nhiên và độ tin cậy của các ứng dụng sử dụng thư viện này. Ngoài ra, thư viện cũng cung cấp các phương thức để thiết lập hạt giống (seed) cho ngẫu nhiên, giúp đảm bảo việc tạo ra các giá trị ngẫu nhiên có thể tái lập được.

* 1. Lớp Car

Thư viện java.util.Random được sử dụng để sinh ra số ngẫu nhiên trong phạm vi cụ thể trong lớp **Car** này. Cụ thể, nó tạo đối tượng rand của lớp Random() và sử dụng để tạo ra vị trí ngẫu nhiên của chiếc xe trên đường, giúp cho vị trí của các chiếc xe xuất hiện trên đường trong trò chơi CarRaceGame được thay đổi liên tục và không đơn điệu. Bằng cách sử dụng phương thức nextInt(), thư viện Random sẽ sinh ra một số nguyên ngẫu nhiên nằm trong phạm vi xác định và gán giá trị này cho biến position\_x của chiếc xe.

while(true) {

position\_x = rand.nextInt(250) + 150;

position\_y = -100;

}

1.2 Lớp Obstacle

Random rand = new Random();

position\_x = rand.nextInt(200) + 150;

Ở đây, hàm nextInt(int n) sẽ sinh ra một số nguyên ngẫu nhiên trong khoảng từ 0 đến 199, và sau đó được cộng thêm 150 để có giá trị của position\_x trong khoảng từ 150 đến 349.

Đồng thời, trong vòng lặp while, đoạn mã sau được sử dụng để sinh ra hai số ngẫu nhiên để sử dụng cho việc sleep của thread:

double randNumOne = Math.random() \* 300 + 200;

double randNumTwo = Math.random() \* 40 + 10;

Ở đây, hàm Math.random() sẽ sinh ra một số thực ngẫu nhiên trong khoảng từ 0 đến 1, sau đó được nhân với một hằng số và cộng với một giá trị để có giá trị của randNumOne và randNumTwo.

Nhờ việc sử dụng các số ngẫu nhiên này, các đối tượng Obstacle sẽ được di chuyển và thay đổi vị trí một cách ngẫu nhiên

1. **Thư viện java.util.logging.Level**

Thư viện java.util.logging.Level trong Java được sử dụng để xác định cấp độ của các thông tin nhật ký (logs) trong ứng dụng. Nó cung cấp một số hằng số được đặt tên cho các mức độ khác nhau của thông tin nhật ký

* 1. Lớp Car

try{

sleep(position\_x \* 5 + 1000);

}catch (InterruptedException ex){

Logger.getLogger(Car.class.getName()).log(Level.SEVERE, null, ex);

}

try{

sleep(position\_x / 20);

}catch (InterruptedException ex){

Logger.getLogger(Car.class.getName()).log(Level.SEVERE,null, ex);

}

Cụ thể trong đoạn code, khi có một ngoại lệ (InterruptedException) xảy ra trong phương thức sleep(), thông điệp lỗi sẽ được ghi vào file log với mức độ SEVERE, tức là mức độ nghiêm trọng nhất. Việc ghi log này giúp cho việc sửa lỗi sau này trở nên dễ dàng hơn.

Thông báo lỗi sẽ cung cấp thông tin chi tiết về ngoại lệ, bao gồm tên của lớp và tên của phương thức tạo ra ngoại lệ

* 1. Lớp Obstacle

try{

sleep((long) (randNumOne / randNumTwo));

}catch (InterruptedException ex){

Logger.getLogger(Car.class.getName()).log(Level.SEVERE,null, ex);

}

Thư viện java.util.logging.Level được sử dụng để ghi log thông tin lỗi nếu quá trình sleep() của thread bị gián đoạn bởi một InterruptedException. Cụ thể, nếu trong quá trình sleep, thread bị ngắt, thì Logger sẽ ghi log một thông báo lỗi với Level là SEVERE và exception được truyền vào trong catch block. Điều này giúp cho việc xử lý lỗi trở nên dễ dàng hơn và cung cấp thông tin hữu ích để giúp người phát triển tìm ra nguyên nhân của lỗi.

1. **Thư viện java.util.logging.Logger**

Thư viện java.util.logging.Logger được sử dụng để ghi log (nhật ký) các hoạt động của ứng dụng. Logger cung cấp các cấp độ nhật ký khác nhau để phân loại các tin nhắn nhật ký theo mức độ quan trọng của chúng, từ cấp độ thấp nhất (FINEST) đến cấp độ cao nhất (SEVERE).

Logger cho phép sử dụng để lưu trữ thông tin gỡ lỗi hoặc để theo dõi và phân tích các hoạt động của ứng dụng.

3.1 Lớp Car

try{

sleep(position\_x \* 5 + 1000);

}catch (InterruptedException ex){

Logger.getLogger(Car.class.getName()).log(Level.SEVERE, null, ex);

}

try{

sleep(position\_x / 20);

}catch (InterruptedException ex){

Logger.getLogger(Car.class.getName()).log(Level.SEVERE,null, ex);

}

Cụ thể trong đoạn code trên,khi một exception (ngoại lệ) InterruptedException xảy ra trong khối try-catch, Logger sẽ ghi lại thông tin về ngoại lệ này cùng với các thông tin khác như tên class, level (mức độ nghiêm trọng của thông báo), v.v. thông qua phương thức getLogger() và log().

3.2 Lớp Obstacle

try{

sleep(position\_x ^3 );

}catch (InterruptedException ex){

Logger.getLogger(Car.class.getName()).log(Level.SEVERE, null, ex);

}

try{

sleep((long) (randNumOne / randNumTwo));

}catch (InterruptedException ex){

Logger.getLogger(Car.class.getName()).log(Level.SEVERE,null, ex);

}

Trong đoạn code này, thư viện java.util.logging.Logger được sử dụng để ghi lại thông tin về lỗi nếu có lỗi xảy ra khi thread đang ngủ. Khi có lỗi xảy ra, Logger sẽ gọi phương thức log() để ghi lại thông tin lỗi và in ra màn hình console hoặc ghi vào tệp log. Level.SEVERE được sử dụng để chỉ định mức độ lỗi là nghiêm trọng nhất, bao gồm các lỗi không thể khôi phục được và sẽ gây ra ảnh hưởng đến hoạt động của chương trình.

**4.Thư viện java.awt.event.ActionEvent và thư viện java.awt.ActionListner**

Thư viện java.awt.event.ActionEvent và java.awt.event.ActionListener cung cấp các công cụ cho phép người dùng tương tác với các thành phần đồ họa trên giao diện người dùng của chương trình.

Cụ thể, java.awt.event.ActionEvent là một sự kiện xảy ra khi người dùng tương tác với một thành phần đồ họa. Khi một sự kiện ActionEvent được kích hoạt, chương trình sẽ tạo ra một đối tượng ActionEvent và gửi nó đến tất cả các ActionListener đã đăng ký với thành phần đó.

java.awt.event.ActionListener là một giao diện được sử dụng để xử lý các sự kiện ActionEvent. Khi một ActionListener được đăng ký với một thành phần đồ họa, nó sẽ cung cấp một phương thức để xử lý sự kiện ActionEvent khi nó xảy ra. Các hành động xử lý này có thể làm bất cứ điều gì mà người lập trình muốn.

b1.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e){

System.exit(0);

}});

b2.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

frame.setVisible(false);

}

});

Trong đoạn code này, thư viện java.awt.event.ActionListener được sử dụng để xử lý các sự kiện khi người dùng tương tác với các button trong giao diện người dùng.

Cụ thể, b1 và b2 là các đối tượng JButton, và addActionListener được sử dụng để gắn các đối tượng ActionListener vào các button đó. Khi button được nhấn, actionPerformed(ActionEvent e) sẽ được gọi để thực hiện một số hành động nhất định.

Trong đoạn code này, khi người dùng nhấn nút "Exit Game", hệ thống sẽ kết thúc ứng dụng bằng cách gọi phương thức System.exit(0). Khi người dùng nhấn nút "Cancel", cửa sổ hiện tại sẽ bị đóng bằng cách gọi phương thức setVisible(false) trên đối tượng JFrame hiện tại.

**5.Thư viện java.awt.event.KeyEvent và thư viện java.event.KeyListener**

Thư viện java.awt.event.KeyEvent cung cấp các lớp và phương thức để xử lý sự kiện phím trên bàn phím trong chương trình Java AWT hoặc Swing. Khi người dùng nhập phím, sự kiện KeyEvent sẽ được tạo ra và gửi đến đối tượng sử lý sự kiện (listener) tương ứng để xử lý.

Các phương thức được cung cấp bởi KeyEvent cho phép truy cập các thông tin liên quan đến sự kiện phím như mã phím, trạng thái của các phím chức năng, các phím điều khiển, ...

Thư viện java.awt.event.KeyListener là một interface cho phép lập trình viên lắng nghe và xử lý các sự kiện liên quan đến phím được bấm trên bàn phím

5.1 Lớp Racing và RaceGame

@Override

public void keyPressed(KeyEvent e) {

if(e.getKeyCode() == VK\_RIGHT && x < 360) {car.setBounds(x += 10, y, 100, 100);}

if(e.getKeyCode() == VK\_LEFT && x > 135) {car.setBounds(x -= 10, y, 100, 100);}

if(e.getKeyCode() == VK\_UP && y > 0) {car.setBounds(x, y -= 10, 100, 100);}

if(e.getKeyCode() == VK\_DOWN && y < 700) {car.setBounds(x, y += 10, 100, 100);}

}

Khi người dùng nhấn một phím mũi tên trên bàn phím, hàm keyPressed được gọi và kiểm tra xem phím nào đã được nhấn. Nếu phím được nhấn là mũi tên phải, x tăng thêm 10 và di chuyển hình ảnh xe sang phải. Nếu phím được nhấn là mũi tên trái, x giảm đi 10 và di chuyển hình ảnh xe sang trái. Nếu phím được nhấn là mũi tên lên, y giảm đi 10 và di chuyển hình ảnh xe lên trên. Nếu phím được nhấn là mũi tên xuống, y tăng thêm 10 và di chuyển hình ảnh xe xuống dưới

setSize(400, 500);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

setResizable(false);

addKeyListener(this);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setVisible(true);

setResizable(false);

setLocation(350,0);

addKeyListener(this);

Trong đoạn code trên, phương thức addKeyListener(this) được sử dụng để đăng ký một KeyListener cho đối tượng hiện tại (trong trường hợp này là JFrame). KeyListener được sử dụng để lắng nghe và xử lý sự kiện phím được nhấn trên bàn phím. Khi một phím được nhấn, sự kiện KeyEvent tương ứng sẽ được tạo ra và gửi đến KeyListener.

1. **Thư viện java.util.ArrayList**

Thư viện java.util.ArrayList cung cấp lớp ArrayList, cho phép lưu trữ và quản lý một danh sách các phần tử trong một mảng động. Các phần tử trong ArrayList có thể được truy cập bằng chỉ số và có thể thêm, xóa hoặc sửa đổi trong quá trình thực thi.

ArrayList<Obstacle> obstacles = new ArrayList();

Obstacle o1 = new Obstacle(obstacle\_one);

Obstacle o2 = new Obstacle(obstacle\_two);

Obstacle o3 = new Obstacle(obstacle\_three);

obstacles.add(o1); obstacles.add(o2); obstacles.add(o3);

Thư viện java.util.ArrayList được sử dụng để tạo danh sách các đối tượng Obstacle. Trong đoạn code này, các đối tượng Obstacle được tạo bằng cách sử dụng constructor Obstacle () và được thêm vào danh sách obstacles thông qua phương thức add (). Sau đó, các đối tượng obstacles được truyền vào đối tượng ThreadControl, mà trong đó ThreadControl sẽ quản lý các Thread được tạo để di chuyển các đối tượng obstacles. Do đó, thư viện java.util.ArrayList được sử dụng để lưu trữ các đối tượng Obstacle và hỗ trợ trong việc quản lý chúng trong trò chơi đua xe.

1. **Thư viện java.swing.\***

Thư viện java.swing.\* là một trong những thư viện quan trọng trong Java để xây dựng các ứng dụng đồ họa (GUI - Graphical User Interface). Nó cung cấp nhiều lớp và giao diện để xây dựng giao diện người dùng như JFrame, JPanel, JButton, JLabel, JTextField, JTable, JList, JMenu, JMenuBar, JToolBar.

JLabel racetrack;

JLabel car;

JLabel obstacle\_one;

JLabel obstacle\_two;

JLabel obstacle\_three;

JTextField points;

JLabel: để hiển thị ảnh nền đường đua, xe và các chướng ngại vật.

JTextField: để hiển thị điểm số.

JFrame: để tạo cửa sổ ứng dụng đua xe.

Ngoài ra

1. **Thư viện java.awt.event.KeyEvent.\***

Thư viện java.awt.event.KeyEvent.\* chứa các hằng số để đại diện cho các sự kiện phím như VK\_UP, VK\_DOWN, VK\_LEFT, VK\_RIGHT, VK\_ENTER, VK\_BACK\_SPACE, VK\_DELETE, VK\_ESCAPE, v.v.

Việc sử dụng các hằng số này có thể giúp dễ dàng xác định phím được nhấn trong một sự kiện phím nhất định

@Override

public void keyPressed(KeyEvent e) {

if(e.getKeyCode() == VK\_RIGHT && x < 360) {car.setBounds(x += 10, y, 100, 100);}

if(e.getKeyCode() == VK\_LEFT && x > 135) {car.setBounds(x -= 10, y, 100, 100);}

if(e.getKeyCode() == VK\_UP && y > 0) {car.setBounds(x, y -= 10, 100, 100);}

if(e.getKeyCode() == VK\_DOWN && y < 900) {car.setBounds(x, y += 10, 100, 100);}}

Đoạn code trên xử lý sự kiện khi người dùng nhấn các phím mũi tên để di chuyển một chiếc xe trên một sân đua. Các phương thức VK\_RIGHT, VK\_LEFT, VK\_UP và VK\_DOWN được sử dụng để xác định phím nào được nhấn. Nếu phím tương ứng được nhấn, vị trí của chiếc xe được điều chỉnh bằng cách sử dụng phương thức setBounds() của đối tượng JLabel.

1. **Thư viện java.lang.Thread.sleep**

Thư viện java.lang.Thread.sleep cung cấp phương thức sleep() để dừng hoạt động của một thread trong một khoảng thời gian nhất định, được chỉ định bởi tham số đầu vào. Khi một thread gọi phương thức sleep(), nó tạm thời ngưng hoạt động trong khoảng thời gian được chỉ định, cho phép các thread khác hoạt động trong khi thread đó đang dừng. Sau khi thời gian ngủ kết thúc, thread đó tiếp tục hoạt động.

while(!racing.judgeStop()){

try {

racing.obstacleMove();

sleep(80);

} catch (InterruptedException ex) {

Logger.getLogger(ThreadControl.class.getName()).log(Level.SEVERE, null, ex);

}

}

Trong đoạn code trên, phương thức sleep() được sử dụng để đảm bảo rằng thread sẽ chuyển động chậm hơn một chút, mỗi lần di chuyển vật cản. Việc này giúp tạo ra hiệu ứng di chuyển của vật cản một cách trơn tru hơn, tránh hiện tượng di chuyển nhanh chóng và không ổn định.