Shape, square

Description automatically generated

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**UNIVERSITY OF TRANSPOST AND COMMUNICATIONS**

Logo

Description automatically generated

**BÁO CÁO BÀI TẬP LỚN**

**MÔN LẬP TRÌNH JAVA**

***ĐỀ TÀI: GAME RẮN SĂN MỒI***

**Giảng viên hướng dẫn: Vũ Huấn**

**Sinh viên thực hiện : Nguyễn Tiến Hoàng 211201040**

**: Phạm Văn Linh 211202119**

**Lớp: CNTT6 – K62**

**Năm 2023**

MỤC LỤC

[**GIỚI THIỆU GAME** 3](#_Toc133883739)

[1. Giới thiệu về game. 3](#_Toc133883740)

[2. Mô tả game. 3](#_Toc133883741)

[**I.** **CÀI ĐẶT GAME** 3](#_Toc133883742)

[1. Class Snake Game 3](#_Toc133883743)

[2. Class Food 12](#_Toc133883744)

[3. Class Barrie 14](#_Toc133883745)

[4. Class Test 15](#_Toc133883746)

[**II.** **GIỚI THIỆU GIAO DIỆN THỰC HIỆN TRÒ CHƠI CHƯƠNG TRÌNH ( HƯỚNG DẪN THỰC HIỆN TRÒ CHƠI )** 17](#_Toc133883747)

[**III.** **KẾT LUẬN VÀ ĐÁNH GIÁ.** 20](#_Toc133883748)

[**IV.** **TÀI LIỆU THAM KHẢO** 20](#_Toc133883749)

**GIỚI THIỆU GAME**

1. Giới thiệu về game.

Game Rắn săn mồi là một trò chơi điện tử cổ điển, được phát triển lần đầu tiên vào những năm 1970. Trong game, người chơi điều khiển một con rắn di chuyển trên màn hình để ăn thức ăn. Mỗi lần ăn được thức ăn, con rắn sẽ dài ra và tốc độ di chuyển cũng sẽ tăng lên. Mục tiêu của trò chơi là đạt được điểm số cao nhất có thể.

Mục đích của bài tập lớn này bọn em làm là để vận dụng các kiến thức đã học để làm một game để hiểu cách lập trình game bằng java và sử dụng công cụ lập trình.

1. Mô tả game.

Game mà bọn em làm ra là tại một thời điểm chỉ có một người chơi , trò sẽ có một màn hình chơi, trong đó người chơi sẽ điều khiển con rắn trên màn hình bằng các phím A( sang trái ) , S (đi xuống ) , D (sang phải) , W ( đi lên ) để ăn đồ ăn để con rắn dài ra . Để dừng trò chơi thì sẽ dùng phím Enter . Khi rắn chết thì sẽ hiện lên điểm và hướng dẫ là “ bấm phím SPACE để bắt đầu lại trò chơi “.

1. **CÀI ĐẶT GAME**

**Game được chia làm 4 class bao gồm :**

* Class Snake Game : Lớp này đại diện cho trò chơi Rắn săn mồi. Nó chứa các biến và phương thức để khởi tạo và điều khiển trò chơi.
* Food: Lớp này đại diện cho thức ăn trong trò chơi. Nó chứa các biến và phương thức để tạo ra thức ăn mới và kiểm tra xem thức ăn có nằm trên rào cản hay trên thân rắn hay không.
* Barrier: Lớp này đại diện cho rào cản trong trò chơi. Nó chứa các biến và phương thức để kiểm tra xem một điểm có nằm trên rào cản hay không và vẽ rào cản lên màn hình.
* test: Lớp này chứa phương thức main() để khởi tạo cửa sổ trò chơi và thêm một đối tượng SnakeGame vào cửa sổ. Ngoài ra còn có một phương thức windowClosing() để lưu điểm số của người chơi khi cửa sổ trò chơi bị đóng.

1. Class Snake Game

**package** snakegame;

**import** java.awt.Color;

**import** java.awt.Dimension;

**import** java.awt.Graphics;

**import** java.awt.Point;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.event.KeyEvent;

**import** java.awt.event.KeyListener;

**import** java.util.ArrayList;

**import** javax.swing.JPanel;

**import** javax.swing.Timer;

**public** **class** SnakeGame **extends** JPanel **implements** ActionListener, KeyListener {

// xác định các biến trò chơi

**private** **static** **final** **int** ***WINDOW\_WIDTH*** = 640;

**private** **static** **final** **int** ***WINDOW\_HEIGHT*** = 480;

**private** **static** **final** **int** ***CELL\_SIZE*** = 10;

**private** **static** **final** **int** ***GAME\_SPEED*** = 100;

**private** **static** **final** **int** ***MAX\_SCORE*** = (***WINDOW\_WIDTH*** / ***CELL\_SIZE***) \* (***WINDOW\_HEIGHT*** / ***CELL\_SIZE***) - 1;

**private** Timer timer;

**private** ArrayList<Point> snakeBody;

**private** Food food;

**private** String direction;

**private** **int** score;

**private** **boolean** gameOver;

**private** ArrayList<Barrier> barriers;

**private** **boolean** paused = **false**;

**public** SnakeGame() {

// khởi tạo các biến trò chơi

setPreferredSize(**new** Dimension(***WINDOW\_WIDTH***, ***WINDOW\_HEIGHT***));

setBackground(Color.***BLACK***);

setFocusable(**true**);

addKeyListener(**this**);

timer = **new** Timer(***GAME\_SPEED***, **this**);

snakeBody = **new** ArrayList<Point>();

snakeBody.add(**new** Point(***WINDOW\_WIDTH*** / 2, ***WINDOW\_HEIGHT*** / 2));

barriers = **new** ArrayList<Barrier>();

barriers.add(**new** Barrier(300, 50, 90, 20, Color.***GRAY***));

barriers.add(**new** Barrier(200, 430, 90, 20, Color.***GRAY***));

food = **new** Food(***WINDOW\_WIDTH***,***WINDOW\_HEIGHT***,***CELL\_SIZE***, barriers, snakeBody);

direction = "RIGHT";

score = 0;

gameOver = **false**;

resetGame();

// bắt đầu vòng lặp trò chơi

timer.start();

grow(3);

}

**public** **void** resetGame() {

snakeBody = **new** ArrayList<Point>();

snakeBody.add(**new** Point(***WINDOW\_WIDTH*** / 2, ***WINDOW\_HEIGHT*** / 2));

food = **new** Food(***WINDOW\_WIDTH***, ***WINDOW\_HEIGHT***, ***CELL\_SIZE***,barriers, snakeBody);

direction = "RIGHT";

score = 0;

gameOver = **false**;

timer.start();

}

@Override

**public** **void** actionPerformed(ActionEvent e) {

// di chuyển con rắn theo hướng hiện tại

Point head = snakeBody.get(0);

Point newHead = **new** Point(head.x, head.y);

**if** (direction.equals("UP")) {

newHead.y -= ***CELL\_SIZE***;

} **else** **if** (direction.equals("DOWN")) {

newHead.y += ***CELL\_SIZE***;

} **else** **if** (direction.equals("LEFT")) {

newHead.x -= ***CELL\_SIZE***;

} **else** **if** (direction.equals("RIGHT")) {

newHead.x += ***CELL\_SIZE***;

}

//check chạm vật cản

**for** (Barrier barrier : barriers) {

**if** (barrier.collidesWith(snakeBody.get(0))) {

gameOver = **true**;

**break**;

}

}

wrapAround(newHead);

// kiểm tra con rắn có ăn ăn thức ăn không

**if** (newHead.equals(food.getLocation())) {

snakeBody.add(0, newHead);

score++;

food.generateFood(***WINDOW\_WIDTH***, ***WINDOW\_HEIGHT***, ***CELL\_SIZE***, barriers, snakeBody);

} **else** {

// di chuyển cơ thể răn và kiểm tra

**if** (snakeBody.contains(newHead)) {

gameOver = **true**;

} **else** {

snakeBody.add(0, newHead);

snakeBody.remove(snakeBody.size() - 1);

}

}

**if** (gameOver == **true**) {

timer.stop();

}

repaint();

}

@Override

**public** **void** keyPressed(KeyEvent e) {

**int** keyCode = e.getKeyCode();

**if** (keyCode == KeyEvent.***VK\_W*** && !direction.equals("DOWN")) {

direction = "UP";

} **else** **if** (keyCode == KeyEvent.***VK\_S*** && !direction.equals("UP")) {

direction = "DOWN";

} **else** **if** (keyCode == KeyEvent.***VK\_A*** && !direction.equals("RIGHT")) {

direction = "LEFT";

} **else** **if** (keyCode == KeyEvent.***VK\_D*** && !direction.equals("LEFT")) {

direction = "RIGHT";

}

**if**(keyCode == KeyEvent.***VK\_SPACE*** && gameOver == **true**){

resetGame();

}

**if**(keyCode == KeyEvent.***VK\_ENTER***){

**if**(paused){

timer.start();

paused = **false**;

}**else**{

timer.stop();

paused = **true**;

}

repaint();

}

}

@Override

**public** **void** keyReleased(KeyEvent e) {}

@Override

**public** **void** keyTyped(KeyEvent e) {}

// private boolean hitWall(Point head) {

// // kiểm tra con rắn nếu đâm vào tường

// return head.x < 0 || head.x >= WINDOW\_WIDTH || head.y < 0 || head.y >= WINDOW\_HEIGHT;

// }

@Override

**protected** **void** paintComponent(Graphics g) {

**super**.paintComponent(g);

// vẽ thân rắn

g.setColor(Color.***GREEN***);

**boolean** isHead = **true**;

**for** (Point p : snakeBody) {

**if** (isHead) {

g.setColor(Color.***pink***);

isHead = **false**;

} **else** {

g.setColor(Color.***GREEN***);

}

g.fillRect(p.x, p.y, ***CELL\_SIZE***, ***CELL\_SIZE***);

}

// vẽ mồi

g.setColor(Color.***RED***);

g.fillRect(food.getLocation().x, food.getLocation().y, ***CELL\_SIZE***, ***CELL\_SIZE***);

//thêm vật cản

**for** (Barrier barrier : barriers) {

barrier.draw(g);

}

// hiển thị điểm

g.setColor(Color.***WHITE***);

g.drawString("Score: " + score, 10, 20);

g.drawString("Pause: Enter", 10, 35);

// thông báo game over

**if** (gameOver == **true**) {

g.drawString("Game Over!", ***WINDOW\_WIDTH*** / 2 - 30, 230);

g.drawString("Score: "+score, ***WINDOW\_WIDTH*** / 2 - 30, 250);

g.drawString("Press space to reset", ***WINDOW\_WIDTH*** / 2 - 30, 270);

timer.stop();

}

**if**(paused){

g.drawString("press enter to continue", ***WINDOW\_WIDTH*** / 2 - 35, 230);

}

}

**public** **int** getScore()

{

**return** **this**.score;

}

**public** **void** grow(**int** amount) {

**for** (**int** i = 0; i < amount; i++) {

Point tail = snakeBody.get(snakeBody.size() - 1);

snakeBody.add(**new** Point(tail.x, tail.y));

}

}

**private** **void** wrapAround(Point head) {

// Kiểm tra xem con rắn có đi ra khỏi màn hình hay không và cập nhật vị trí của nó

**if** (head.x < 0) {

head.x = ***WINDOW\_WIDTH*** - ***CELL\_SIZE***;

} **else** **if** (head.x >= ***WINDOW\_WIDTH***) {

head.x = 0;

} **else** **if** (head.y < 0) {

head.y = ***WINDOW\_HEIGHT*** - ***CELL\_SIZE***;

} **else** **if** (head.y >= ***WINDOW\_HEIGHT***) {

head.y = 0;

}

}

}

* Các biến:
* WINDOW\_WIDTH, WINDOW\_HEIGHT, CELL\_SIZE, GAME\_SPEED, MAX\_SCORE: Đây là các hằng số để xác định kích thước cửa sổ trò chơi, kích thước của mỗi ô vuông, tốc độ trò chơi và điểm số tối đa.
* timer: Đối tượng Timer để điều khiển vòng lặp trò chơi.
* snakeBody: Danh sách các điểm đại diện cho thân rắn.
* food: Đối tượng Food đại diện cho thức ăn trong trò chơi.
* direction: Chuỗi để xác định hướng di chuyển hiện tại của con rắn.
* score: Biến để lưu điểm số của người chơi.
* gameOver: Biến boolean để xác định xem trò chơi có kết thúc hay không.
* barriers: Danh sách các đối tượng Barrier đại diện cho các rào cản trong trò chơi.
* paused: Biến boolean để xác định xem trò chơi có bị tạm dừng hay không.
* Các phương thức:
* SnakeGame(): Constructor của lớp. Trong constructor này, các biến trò chơi được khởi tạo và trò chơi được bắt đầu với việc khởi tạo các đối tượng cần thiết như thức ăn và rào cản.
* resetGame(): Phương thức này được sử dụng để khởi động lại trò chơi. Nó thiết lập lại các biến trò chơi về giá trị ban đầu và khởi tạo lại các đối tượng cần thiết như thức ăn và rào cản.
* actionPerformed(ActionEvent e): Phương thức này được gọi mỗi khi timer phát ra một sự kiện. Nó xử lý sự kiện di chuyển con rắn theo hướng hiện tại và kiểm tra xem con rắn có ăn thức ăn hay không. Nếu con rắn ăn được thức ăn, điểm số sẽ được cộng thêm và thức ăn mới sẽ được tạo ra. Nếu con rắn tự cắn vào thân mình hoặc đâm vào rào cản, trò chơi sẽ kết thúc.
* keyPressed(KeyEvent e): Phương thức này được gọi khi người dùng nhấn một phím. Nó xử lý sự kiện nhấn phím và thay đổi hướng di chuyển của con rắn theo phím mà người dùng nhấn. Ngoài ra, nếu người dùng nhấn phím Space khi trò chơi kết thúc, trò chơi sẽ được khởi động lại. Nếu người dùng nhấn phím Enter, trò chơi sẽ được tạm dừng hoặc tiếp tục.
* paintComponent(Graphics g): Phương thức này được gọi mỗi khi cửa sổ trò chơi cần được vẽ lại. Nó vẽ thân rắn, thức ăn và các đối tượng khác trong trò chơi lên màn hình. Ngoài ra, nó cũng hiển thị điểm số của người chơi và thông báo game over khi rắn chết và ghi hướng dẫn nhấn Space để reset lại game . Còn nếu người chơi nhấn Enter thì nó sẽ hiển thị hướng dẫn Enter để game hoạt động tiếp.
* grow(**int** amount) : phương thức này dùng để tăng độ dài thân của rắn với biến int amount là dộ dài thân mà bạn muốn thêm trừ đầu rắn (ví dụ grow(3) thì thân rắn sẽ dài là 3)
* wrapAround(Point head) : phương thức này dùng để con rắn có thể đi ra khỏi phần viền giao diện và xuất diện ở phía đối diện.

1. Class Food

**package** snakegame;

**import** java.awt.Point;

**import** java.util.List;

**import** java.util.Random;

//import snakegame.Barrier;

**public** **class** Food {

**private** Point location;

**private** Random random;

**public** Food(**int** windowWidth, **int** windowHeight, **int** cellSize, List<Barrier> barriers, List<Point> snakeBody) {

location = **new** Point();

random = **new** Random();

generateFood(windowWidth, windowHeight, cellSize, barriers, snakeBody);

}

**public** **void** generateFood(**int** windowWidth, **int** windowHeight, **int** cellSize, List<Barrier> barriers, List<Point> snakeBody) {

**int** x, y;

**do** {

x = random.nextInt(windowWidth / cellSize) \* cellSize;

y = random.nextInt(windowHeight / cellSize) \* cellSize;

location.setLocation(x, y);

} **while** (isOnBarrier(barriers) && isOnSnake(snakeBody));

}

**public** Point getLocation() {

**return** location;

}

**private** **boolean** isOnBarrier(List<Barrier> barriers) {

**for** (Barrier barrier : barriers) {

**if** (barrier.collidesWith(location)) {

**return** **true**;

}

}

**return** **false**;

}

**private** **boolean** isOnSnake(List<Point> snakeBody) {

**for** (Point point : snakeBody) {

**if** (point.equals(location)) {

**return** **true**;

}

}

**return** **false**;

}

}

* Các biến:
* location: Đối tượng Point để lưu vị trí của thức ăn.
* random: Đối tượng Random để tạo ra vị trí ngẫu nhiên cho thức ăn mới.
* Các phương thức:
* Food(int windowWidth, int windowHeight, int cellSize, List<Barrier> barriers, List<Point> snakeBody): Constructor của lớp. Trong constructor này, các biến được khởi tạo và thức ăn mới được tạo ra.
* generateFood(int windowWidth, int windowHeight, int cellSize, List<Barrier> barriers, List<Point> snakeBody): Phương thức này được sử dụng để tạo ra thức ăn mới cho con rắn. Nó sử dụng đối tượng Random để tạo ra vị trí ngẫu nhiên cho thức ăn mới và kiểm tra xem thức ăn có nằm trên rào cản hay trên thân rắn hay không. Nếu có, thức ăn mới sẽ được tạo ra ở vị trí khác.
* getLocation(): Phương thức này trả về vị trí hiện tại của thức ăn.
* isOnBarrier(List<Barrier> barriers): Phương thức này kiểm tra xem thức ăn có nằm trên rào cản hay không.
* isOnSnake(List<Point> snakeBody): Phương thức này kiểm tra xem thức ăn có nằm trên thân rắn hay không.

1. Class Barrie

package snakegame;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.Point;

import java.awt.Rectangle;

public class Barrier {

private Rectangle rect;

private Color color;

public Barrier(int x, int y, int width, int height, Color color) {

rect = new Rectangle(x, y, width, height);

this.color = color;

}

public boolean collidesWith(Point p) {

return rect.contains(p);

}

public void draw(Graphics g) {

g.setColor(color);

g.fillRect(rect.x, rect.y, rect.width, rect.height);

}

}

* Các biến:
* rect: Đối tượng Rectangle để lưu vị trí và kích thước của rào cản.
* color: Đối tượng Color để lưu màu sắc của rào cản.
* Các phương thức:
* Barrier(int x, int y, int width, int height, Color color): Constructor của lớp. Trong constructor này, các biến được khởi tạo và đối tượng Rectangle được tạo ra để lưu vị trí và kích thước của rào cản.
* collidesWith(Point p): Phương thức này kiểm tra xem một điểm có nằm trên rào cản hay không.
* draw(Graphics g): Phương thức này vẽ rào cản lên màn hình.

1. Class Test

package snakegame;

import java.awt.CardLayout;

import java.awt.event.WindowAdapter;

import java.awt.event.WindowEvent;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import javax.swing.JFrame;

public class test {

public static void main(String[] args) {

// thiết lập của sổ trò chơi

JFrame frame = new JFrame("Snake Game");

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setResizable(false);

SnakeGame t = new SnakeGame();

frame.add(t);

frame.pack();

frame.setLocationRelativeTo(null);

// Khai báo đường dẫn và tên file để lưu dữ liệu

String filePath = "data.txt";

frame.addWindowListener(new WindowAdapter() {

@Override

public void windowClosing(WindowEvent e) {

// In ra giá trị khi frame bị tắt

System.out.println(t.getScore());

try {

// Khởi tạo một đối tượng BufferedWriter để ghi dữ liệu vào file

BufferedWriter writer = new BufferedWriter(new FileWriter(filePath));

// Ghi dữ liệu vào file

writer.write("Score: "+t.getScore());

// Đóng đối tượng BufferedWriter để lưu dữ liệu vào file

writer.close();

} catch (IOException ex) {

ex.printStackTrace();

}

}

});

frame.setVisible(true);

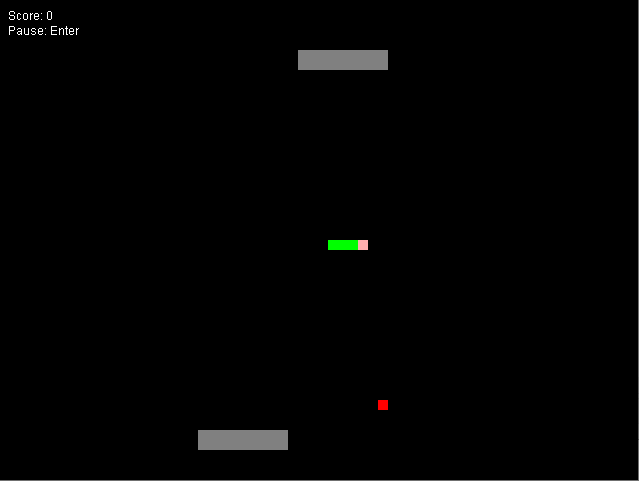
}

}

* Các biến:
* Không có biến nào trong lớp này.
* Các phương thức:
* main(String[] args): Đây là phương thức chính của lớp. Nó khởi tạo cửa sổ trò chơi và thêm một đối tượng SnakeGame vào cửa sổ. Ngoài ra còn có một phương thức windowClosing() để lưu điểm số của người chơi khi cửa sổ trò chơi bị đóng.

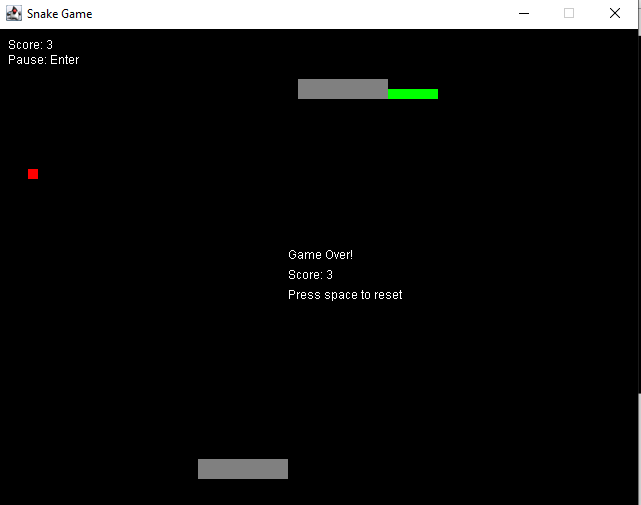
1. **GIỚI THIỆU GIAO DIỆN THỰC HIỆN TRÒ CHƠI CHƯƠNG TRÌNH ( HƯỚNG DẪN THỰC HIỆN TRÒ CHƠI )**

* Khi ta bấm chạy chương trình thì giao diện trò chơi xuất hiện với con rắn đang ở giữa và di chuyển theo chiều ngang từ trái qua phải.Ở góc trên cùng bên trái có mục score ghi điểm được cập nhật khi rắn ăn mồi và mục hướng dẫn tạm dừng khi nhấn nút Enter.

****

* Khi muốn rắn di chuyển hướng lên xuống trái phải ta ấn lần lượt các nút W , S , A , D .Khi dừng ta ấn Enter sẽ hiện dòng chữ “ press enter to continue . Khi rắn đâm vào tường sẽ hiện dòng chữ “Game over” , điểm và “press space to reset “ . Nếu ta bấm Space thì game sẽ bắt đầu lại từ đầu.

****

****

1. **KẾT LUẬN VÀ ĐÁNH GIÁ.**
2. Ưu điểm

Có đủ các chức năng cần thiết của game. Game chạy tốt không gặp lỗi

1. Nhược điểm

Giao diện chưa được đẹp mắt. Chưa sử dụng MySQL để lưu điểm.

1. Hướng phát triển

Thêm phần menu cho game gồm nút Start , phần chọn các level khác nhau , giao diện đẹp hơn , lồng thêm âm thanh hình ảnh và đưa ra danh sách các điểm cao nhất bằng MySQL.

1. **TÀI LIỆU THAM KHẢO**