Game.java

import java.util\*;

import javax.imageio.ImageIO;

import java.util.Timer;

import java.awt.\*;

import java.awt.event.\*;

import java.awt.image.\*;

import java.io.\*;

import javax.swing.\*;

class Game extends JPanel {

private Timer timer;

private Snake snake;

private Point cherry;

private int points = 0;

private int best = 0;

private BufferedImage image;

private GameStatus status;

private boolean didLoadCherryImage = true;

private static Font FONT\_M = new Font("MV Boli", Font.PLAIN, 24);

private static Font FONT\_M\_ITALIC = new Font("MV Boli", Font.ITALIC, 24);

private static Font FONT\_L = new Font("MV Boli", Font.PLAIN, 84);

private static Font FONT\_XL = new Font("MV Boli", Font.PLAIN, 150);

private static int WIDTH = 760;

private static int HEIGHT = 520;

private static int DELAY = 50;

public Game() {

try {

image = ImageIO.read(new File("cherry.png"));

} catch (IOException e) {

didLoadCherryImage = false;

}

addKeyListener(new KeyListener());

setFocusable(true);

setBackground(new Color(130, 205, 71));

setDoubleBuffered(true);

snake = new Snake(WIDTH / 2, HEIGHT / 2);

status = GameStatus.NOT\_STARTED;

repaint();

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

render(g);

Toolkit.getDefaultToolkit().sync();

}

private void update() {

snake.move();

if (cherry != null && snake.getHead().intersects(cherry, 20)) {

snake.addTail();

cherry = null;

points++;

}

if (cherry == null) {

spawnCherry();

}

checkForGameOver();

}

private void reset() {

points = 0;

cherry = null;

snake = new Snake(WIDTH / 2, HEIGHT / 2);

setStatus(GameStatus.RUNNING);

}

private void setStatus(GameStatus newStatus) {

switch(newStatus) {

case RUNNING:

timer = new Timer();

timer.schedule(new GameLoop(), 0, DELAY);

break;

case PAUSED:

timer.cancel();

case GAME\_OVER:

timer.cancel();

best = points > best ? points : best;

break;

}

status = newStatus;

}

private void togglePause() {

setStatus(status == GameStatus.PAUSED ? GameStatus.RUNNING : GameStatus.PAUSED);

}

private void checkForGameOver() {

Point head = snake.getHead();

boolean hitBoundary = head.getX() <= 20

|| head.getX() >= WIDTH + 10

|| head.getY() <= 40

|| head.getY() >= HEIGHT + 30;

boolean ateItself = false;

for(Point t : snake.getTail()) {

ateItself = ateItself || head.equals(t);

}

if (hitBoundary || ateItself) {

setStatus(GameStatus.GAME\_OVER);

}

}

public void drawCenteredString(Graphics g, String text, Font font, int y) {

FontMetrics metrics = g.getFontMetrics(font);

int x = (WIDTH - metrics.stringWidth(text)) / 2;

g.setFont(font);

g.drawString(text, x, y);

}

private void render(Graphics g) {

Graphics2D g2d = (Graphics2D) g;

g2d.setColor(Color.BLACK);

g2d.setFont(FONT\_M);

if (status == GameStatus.NOT\_STARTED) {

drawCenteredString(g2d, "SNAKE", FONT\_XL, 200);

drawCenteredString(g2d, "GAME", FONT\_XL, 300);

drawCenteredString(g2d, "Press any key to begin", FONT\_M\_ITALIC, 330);

return;

}

Point p = snake.getHead();

g2d.drawString("SCORE: " + String.format ("%02d", points), 20, 30);

g2d.drawString("BEST: " + String.format ("%02d", best), 630, 30);

if (cherry != null) {

if (didLoadCherryImage) {

g2d.drawImage(image, cherry.getX(), cherry.getY(), 60, 60, null);

} else {

g2d.setColor(Color.BLACK);

g2d.fillOval(cherry.getX(), cherry.getY(), 10, 10);

g2d.setColor(Color.BLACK);

}

}

if (status == GameStatus.GAME\_OVER) {

drawCenteredString(g2d, "Press enter to start again", FONT\_M\_ITALIC, 330);

drawCenteredString(g2d, "GAME OVER", FONT\_L, 300);

}

if (status == GameStatus.PAUSED) {

g2d.drawString("Paused", 600, 14);

}

g2d.setColor(new Color(33, 70, 199));

g2d.fillRect(p.getX(), p.getY(), 10, 10);

for(int i = 0, size = snake.getTail().size(); i < size; i++) {

Point t = snake.getTail().get(i);

g2d.fillRect(t.getX(), t.getY(), 10, 10);

}

g2d.setColor(Color.RED);

g2d.setStroke(new BasicStroke(4));

g2d.drawRect(20, 40, WIDTH, HEIGHT);

}

public void spawnCherry() {

cherry = new Point((new Random()).nextInt(WIDTH - 60) + 20,

(new Random()).nextInt(HEIGHT - 60) + 40);

}

private class KeyListener extends KeyAdapter {

@Override

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();

if (status == GameStatus.RUNNING) {

switch(key) {

case KeyEvent.VK\_LEFT: snake.turn(Direction.LEFT); break;

case KeyEvent.VK\_RIGHT: snake.turn(Direction.RIGHT); break;

case KeyEvent.VK\_UP: snake.turn(Direction.UP); break;

case KeyEvent.VK\_DOWN: snake.turn(Direction.DOWN); break;

}

}

if (status == GameStatus.NOT\_STARTED) {

setStatus(GameStatus.RUNNING);

}

if (status == GameStatus.GAME\_OVER && key == KeyEvent.VK\_ENTER) {

reset();

}

if (key == KeyEvent.VK\_P) {

togglePause();

}

}

}

private class GameLoop extends java.util.TimerTask {

public void run() {

update();

repaint();

}

}

}

enum GameStatus

{

NOT\_STARTED, RUNNING, PAUSED, GAME\_OVER

}

enum Direction {

UP, DOWN, LEFT, RIGHT;

public boolean isX() {

return this == LEFT || this == RIGHT;

}

public boolean isY() {

return this == UP || this == DOWN;

}

}

class Point {

private int x;

private int y;

public Point(int x, int y) {

this.x = x;

this.y = y;

}

public Point(Point p) {

this.x = p.getX();

this.y = p.getY();

}

public void move(Direction d, int value) {

switch(d) {

case UP: this.y -= value; break;

case DOWN: this.y += value; break;

case RIGHT: this.x += value; break;

case LEFT: this.x -= value; break;

}

}

public int getX() {

return x;

}

public int getY() {

return y;

}

public Point setX(int x) {

this.x = x;

return this;

}

public Point setY(int y) {

this.y = y;

return this;

}

public boolean equals(Point p) {

return this.x == p.getX() && this.y == p.getY();

}

public String toString() {

return "(" + x + ", " + y + ")";

}

public boolean intersects(Point p) {

return intersects(p, 10);

}

public boolean intersects(Point p, int tolerance) {

int diffX = Math.abs(x - p.getX());

int diffY = Math.abs(y - p.getY());

return this.equals(p) || (diffX <= tolerance && diffY <= tolerance);

}

}

class Snake {

private Direction direction;

private Point head;

private ArrayList<Point> tail;

public Snake(int x, int y) {

this.head = new Point(x, y);

this.direction = Direction.RIGHT;

this.tail = new ArrayList<Point>();

this.tail.add(new Point(0, 0));

this.tail.add(new Point(0, 0));

this.tail.add(new Point(0, 0));

}

public void move() {

ArrayList<Point> newTail = new ArrayList<Point>();

for (int i = 0, size = tail.size(); i < size; i++) {

Point previous = i == 0 ? head : tail.get(i - 1);

newTail.add(new Point(previous.getX(), previous.getY()));

}

this.tail = newTail;

this.head.move(this.direction, 10);

}

public void addTail() {

this.tail.add(new Point(-10, -10));

}

public void turn(Direction d) {

if (d.isX() && direction.isY() || d.isY() && direction.isX()) {

direction = d;

}

}

public ArrayList<Point> getTail() {

return this.tail;

}

public Point getHead() {

return this.head;

}

}

Main.java

import java.awt.EventQueue;

import javax.swing.JFrame;

public class Main extends JFrame {

public Main() {

initUI();

}

private void initUI() {

add(new Game());

setTitle("Snake");

setSize(800, 610);

setLocationRelativeTo(null);

setResizable(false);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

EventQueue.invokeLater(() -> {

Main ex = new Main();

ex.setVisible(true);

});

}

}

Output:

Text

Description automatically generated

Chart

Description automatically generated

Text, logo

Description automatically generated with medium confidence