Nikhil V., Timon S., Hanlin S., Anurag K.

CIS 043

Professor Mortezaie

19 July 2016

Snake Game

**Snake.Java:**

//Creating the snake

public class Snake {

// Stores the joints / body part locations for our snake in arrays

private final int[] x = new int[Board.getAllDots()];

private final int[] y = new int[Board.getAllDots()];

// Stores direction of our snake

private boolean movingLeft = false;

private boolean movingRight = false;

private boolean movingUp = false;

private boolean movingDown = false;

private int joints = 0; // Stores # of dots / joints the snake has (starts

// with 3)

//Set and get methods for the location of the snake

public int getSnakeX(int index) {

return x[index];

}

public int getSnakeY(int index) {

return y[index];

}

public void setSnakeX(int i) {

x[0] = i;

}

public void setSnakeY(int i) {

y[0] = i;

}

//Set and get methods for the movement of the snake

public boolean isMovingLeft() {

return movingLeft;

}

public void setMovingLeft(boolean movingLeft) {

this.movingLeft = movingLeft;

}

public boolean isMovingRight() {

return movingRight;

}

public void setMovingRight(boolean movingRight) {

this.movingRight = movingRight;

}

public boolean isMovingUp() {

return movingUp;

}

public void setMovingUp(boolean movingUp) {

this.movingUp = movingUp;

}

public boolean isMovingDown() {

return movingDown;

}

public void setMovingDown(boolean movingDown) {

this.movingDown = movingDown;

}

//Set and get methods for the joints of the snake

public int getJoints() {

return joints;

}

public void setJoints(int j) {

joints = j;

}

//Method for how the snake moves

public void move() {

//Continues to loop for every joint the snake has

for (int i = joints; i > 0; i--) {

// Moves the joints of the snake 'up the chain'

// Meaning, the joint of the snake all move up one

x[i] = x[(i - 1)];

y[i] = y[(i - 1)];

}

// Moves snake to the left

if (movingLeft) {

x[0] -= Board.getDotSize();

}

// To the right

if (movingRight) {

x[0] += Board.getDotSize();

}

// Down

if (movingDown) {

y[0] += Board.getDotSize();

}

// And finally up

if (movingUp) {

y[0] -= Board.getDotSize();

}

// Dotsize represents the size of the joint, so a pixel of DOTSIZE (25)

// gets added on to the snake in that direction

}

}

----------------------------------------------------------------------------------------------------------------------------

**Board.Java:**

//Creating the board for the game

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.FontMetrics;

import java.awt.Graphics;

import java.awt.Toolkit;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.KeyAdapter;

import java.awt.event.KeyEvent;

import javax.swing.JPanel;

import javax.swing.Timer;

import java.security.SecureRandom;

public class Board extends JPanel implements ActionListener {

// Holds height and width of the window

private final static int BOARDWIDTH = 1000;

private final static int BOARDHEIGHT = 980;

// Used to represent pixel size of food & our snake's joints

private final static int PIXELSIZE = 25;

// The total amount of pixels the game could possibly have.

// We don't want less, because the game would end prematurely.

// We don't more because there would be no way to let the player win.

private final static int TOTALPIXELS = (BOARDWIDTH \* BOARDHEIGHT)

/ (PIXELSIZE \* PIXELSIZE);

//Random number generator used later

SecureRandom randomNumbers = new SecureRandom();

// Check to see if the game is running

private boolean inGame = true;

// Timer used to record tick times

private Timer timer;

// Used to set game speed, the lower the #, the faster the snake travels

// which in turn makes the game harder.

private static int speed = 45;

// Instances of our snake & food so we can use their methods

private Snake snake = new Snake();

private Food food = new Food();

public Board() {

addKeyListener(new Keys());

setBackground(Color.BLACK);

setFocusable(true);

setPreferredSize(new Dimension(BOARDWIDTH, BOARDHEIGHT));

initializeGame();

}

// Used to paint our components to the screen

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

draw(g);

}

// Draw our Snake & Food (Called on repaint()).

void draw(Graphics g) {

int x = randomNumbers.nextInt(6);

// Only draw if the game is running / the snake is alive

if (inGame == true) {

switch (x){ // Make food rainbow

case 0: g.setColor(Color.PINK);

break;

case 1: g.setColor(Color.BLUE);

break;

case 2: g.setColor(Color.RED);

break;

case 3: g.setColor(Color.GREEN);

break;

case 4: g.setColor(Color.WHITE);

break;

case 5: g.setColor(Color.ORANGE);

break;

case 6: g.setColor(Color.YELLOW);

break;

}

g.fillRect(food.getFoodX(), food.getFoodY(), PIXELSIZE, PIXELSIZE); // food

// Draw our snake.

for (int i = 0; i < snake.getJoints(); i++) {

// Snake's head

if (i == 0) {

g.setColor(Color.GREEN);

g.fillRect(snake.getSnakeX(i), snake.getSnakeY(i),

PIXELSIZE, PIXELSIZE);

// Body of snake

} else {

g.fillRect(snake.getSnakeX(i), snake.getSnakeY(i),

PIXELSIZE, PIXELSIZE);

}

}

// Sync our graphics together

Toolkit.getDefaultToolkit().sync();

} else {

// If we're not alive, then we end our game

endGame(g);

}

}

void initializeGame() {

snake.setJoints(3); // set our snake's initial size

// Create our snake's body

for (int i = 0; i < snake.getJoints(); i++) {

snake.setSnakeX(BOARDWIDTH / 2);

snake.setSnakeY(BOARDHEIGHT / 2);

}

// Start off our snake moving right

snake.setMovingRight(true);

// Generate our first 'food'

food.createFood();

// set the timer to record our game's speed / make the game move

timer = new Timer(speed, this);

timer.start();

}

// if our snake is in the close proximity of the food..

void checkFoodCollisions() {

if ((proximity(snake.getSnakeX(0), food.getFoodX(), 20))

&& (proximity(snake.getSnakeY(0), food.getFoodY(), 20))) {

// Add a 'joint' to our snake

snake.setJoints(snake.getJoints() + 1);

// Create new food

food.createFood();

food.getScore();

}

}

// Used to check collisions with snake's self and board edges

void checkCollisions() {

// If the snake hits its' own joints..

for (int i = snake.getJoints(); i > 0; i--) {

// Snake cant intersect with itself if it's not larger than 5

if ((i > 5)

&& (snake.getSnakeX(0) == snake.getSnakeX(i) && (snake

.getSnakeY(0) == snake.getSnakeY(i)))) {

inGame = false; // then the game ends

}

}

// If the snake intersects with the board edges..

if (snake.getSnakeY(0) >= BOARDHEIGHT) {

inGame = false;

}

if (snake.getSnakeY(0) < 0) {

inGame = false;

}

if (snake.getSnakeX(0) >= BOARDWIDTH) {

inGame = false;

}

if (snake.getSnakeX(0) < 0) {

inGame = false;

}

// If the game has ended, then we can stop our timer

if (!inGame) {

timer.stop();

}

}

void endGame(Graphics g) {

// Create a message telling the player the game is over

String message = "Game over. Press enter to play again.";

String message2 = "Total score is: " + (food.getScore() - 5);

food.setScore(0);

// Create a new font instance

Font font = new Font("Serif", Font.ROMAN\_BASELINE, 30);

FontMetrics metrics = getFontMetrics(font);

Font scorefont = new Font("Serif", Font.ROMAN\_BASELINE, 30);

FontMetrics scoremetrics = getFontMetrics(scorefont);

// Set the color of the text to red, and set the font

g.setColor(Color.cyan);

g.setFont(font);

g.drawString(message, (BOARDWIDTH - metrics.stringWidth(message)) / 2,

BOARDHEIGHT / 3);

g.setColor(Color.cyan);

g.setFont(scorefont);

// Draw the message to the board

g.drawString(message2, (BOARDWIDTH - scoremetrics.stringWidth(message2)) / 2, BOARDHEIGHT / 2);

System.out.println("Game Ended");

}

// Run constantly as long as we're in game.

@Override

public void actionPerformed(ActionEvent e) {

if (inGame == true) {

checkFoodCollisions();

checkCollisions();

snake.move();

System.out.println(snake.getSnakeX(0) + " " + snake.getSnakeY(0)

+ " " + food.getFoodX() + ", " + food.getFoodY());

}

// Repaint or 'render' our screen

repaint();

}

private class Keys extends KeyAdapter {

//class Keys reads and takes actions from keyboard input

@Override

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();

if ((key == KeyEvent.VK\_LEFT) && (!snake.isMovingRight())) {

snake.setMovingLeft(true);

snake.setMovingUp(false);

snake.setMovingDown(false);

}

if ((key == KeyEvent.VK\_RIGHT) && (!snake.isMovingLeft())) {

snake.setMovingRight(true);

snake.setMovingUp(false);

snake.setMovingDown(false);

}

if ((key == KeyEvent.VK\_UP) && (!snake.isMovingDown())) {

snake.setMovingUp(true);

snake.setMovingRight(false);

snake.setMovingLeft(false);

}

if ((key == KeyEvent.VK\_DOWN) && (!snake.isMovingUp())) {

snake.setMovingDown(true);

snake.setMovingRight(false);

snake.setMovingLeft(false);

}

if ((key == KeyEvent.VK\_ENTER) && (inGame == false)) {

inGame = true;

snake.setMovingDown(false);

snake.setMovingRight(false);

snake.setMovingLeft(false);

snake.setMovingUp(false);

initializeGame();

}

}

}

private boolean proximity(int a, int b, int closeness) {

return Math.abs((long) a - b) <= closeness;

}

public static int getAllDots() {

return TOTALPIXELS;

}

public static int getDotSize() {

return PIXELSIZE;

}

}

----------------------------------------------------------------------------------------------------------------------------

**Game.Java:**

//Main

import java.awt.EventQueue;

import javax.swing.JFrame;

public class Game extends JFrame {

//Creates the game

Game() {

add(new Board()); //creates the board

setResizable(false);

pack();

setTitle("Snake");

setLocationRelativeTo(null);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

// Creates a new thread so our GUI can process itself

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

JFrame frame = new Game();

frame.setVisible(true);

}

});

}

}

--------------------------------------------------------------------------------------------------------------------------------

**Food.Java:**

//Creating the food

public class Food {

private Snake snake = new Snake();

private int foodX; // Stores X position of our food

private int foodY; // Stores Y position of our food

private int score; // Stores the score

// Used to determine random position of food

private final int RANDOMPOSITION = 40;

public void createFood() {

score += 5; //Each food is worth five points

// Set our food's x & y position to a random position

int location = (int) (Math.random() \* RANDOMPOSITION); //Creates random pixel location for X

foodX = ((location \* Board.getDotSize())); //Location multiplied by pixel size

location = (int) (Math.random() \* RANDOMPOSITION); //For Y

foodY = ((location \* Board.getDotSize()));

if ((foodX == snake.getSnakeX(0)) && (foodY == snake.getSnakeY(0))) {

createFood(); //If the snake touches the food, create a new piece of food

}

}

//Set and get methods for the score

public void setScore(int score){

this.score = score;

}

public int getScore(){

return score;

}

//Get methods for food location (x & y)

public int getFoodX() {

return foodX;

}

public int getFoodY() {

return foodY;

}

}

--------------------------------------------------------------------------------------------------------------------------------

**Game.Java:**

package ch10;  **// declare package name as “ch10”**

import java.awt.EventQueue; **//import The Java Abstract Window Toolkit Event Pa**

**ckage(GUI)**

import javax.swing.JFrame; **//import The Java Swing GUI Components Package (GUI)**

public class Game extends JFrame { **//declare class and open the results of the program in a JFrame**

Game() {  **// Creates method inside “Game” class**

add(new Board()); **//Creates a “new board” inside game as defined in Board class**

setResizable(false); **//Sets the parameter of method setResizable to essentially “cannot resize”**

pack(); **//store this code in “game” in allowable parameters**

setTitle("Snake"); **//Set the title of the game to be shown on “marquee” as “Snake” in game**

setLocationRelativeTo(null); **//location having no relative measure**

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); **//when gamer exits code, close the JFrame**

}

public static void main(String[] args) {  **//declare method main**

EventQueue.invokeLater(new Runnable() { **// Creates a new thread so our GUI can process itself**

@Override

public void run() { **//run method**

JFrame frame = new Game(); **//Whenever we open a new JFrame, start a new Game (refers back to whatever code is inside the “Game” method**

frame.setVisible(true); **//allows the gamer to be able to see each frame**

}

});

}

}

-------------------------------------------------------------------------------------------------------------------------------