**Snake**

In this part of the Java 2D games tutorial, we will create a Java Snake game clone.

**Snake**

*Snake* is an older classic video game. It was first created in late 70s. Later it was brought to PCs. In this game the player controls a snake. The objective is to eat as many apples as possible. Each time the snake eats an apple, its body grows. The snake must avoid the walls and its own body. This game is sometimes called *Nibbles*.

**Development**

The size of each of the joints of a snake is 10px. The snake is controlled with the cursor keys. Initially, the snake has three joints. If the game is finished, the "Game Over" message is displayed in the middle of the board.

Board.java

package com.zetcode;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.FontMetrics;

import java.awt.Graphics;

import java.awt.Image;

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.ImageIcon;

import javax.swing.JPanel;

import javax.swing.Timer;

public class Board extends JPanel implements ActionListener {

private final int B\_WIDTH = 300;

private final int B\_HEIGHT = 300;

private final int DOT\_SIZE = 10;

private final int ALL\_DOTS = 900;

private final int RAND\_POS = 29;

private final int DELAY = 140;

private final int x[] = new int[ALL\_DOTS];

private final int y[] = new int[ALL\_DOTS];

private int dots;

private int apple\_x;

private int apple\_y;

private boolean leftDirection = false;

private boolean rightDirection = true;

private boolean upDirection = false;

private boolean downDirection = false;

private boolean inGame = true;

private Timer timer;

private Image ball;

private Image apple;

private Image head;

public Board() {

addKeyListener(new TAdapter());

setBackground(Color.black);

setFocusable(true);

setPreferredSize(new Dimension(B\_WIDTH, B\_HEIGHT));

loadImages();

initGame();

}

private void loadImages() {

ImageIcon iid = new ImageIcon("dot.png");

ball = iid.getImage();

ImageIcon iia = new ImageIcon("apple.png");

apple = iia.getImage();

ImageIcon iih = new ImageIcon("head.png");

head = iih.getImage();

}

private void initGame() {

dots = 3;

for (int z = 0; z < dots; z++) {

x[z] = 50 - z \* 10;

y[z] = 50;

}

locateApple();

timer = new Timer(DELAY, this);

timer.start();

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

doDrawing(g);

}

private void doDrawing(Graphics g) {

if (inGame) {

g.drawImage(apple, apple\_x, apple\_y, this);

for (int z = 0; z < dots; z++) {

if (z == 0) {

g.drawImage(head, x[z], y[z], this);

} else {

g.drawImage(ball, x[z], y[z], this);

}

}

Toolkit.getDefaultToolkit().sync();

} else {

gameOver(g);

}

}

private void gameOver(Graphics g) {

String msg = "Game Over";

Font small = new Font("Helvetica", Font.BOLD, 14);

FontMetrics metr = getFontMetrics(small);

g.setColor(Color.white);

g.setFont(small);

g.drawString(msg, (B\_WIDTH - metr.stringWidth(msg)) / 2, B\_HEIGHT / 2);

}

private void checkApple() {

if ((x[0] == apple\_x) && (y[0] == apple\_y)) {

dots++;

locateApple();

}

}

private void move() {

for (int z = dots; z > 0; z--) {

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

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

}

if (leftDirection) {

x[0] -= DOT\_SIZE;

}

if (rightDirection) {

x[0] += DOT\_SIZE;

}

if (upDirection) {

y[0] -= DOT\_SIZE;

}

if (downDirection) {

y[0] += DOT\_SIZE;

}

}

private void checkCollision() {

for (int z = dots; z > 0; z--) {

if ((z > 4) && (x[0] == x[z]) && (y[0] == y[z])) {

inGame = false;

}

}

if (y[0] >= B\_HEIGHT) {

inGame = false;

}

if (y[0] < 0) {

inGame = false;

}

if (x[0] >= B\_WIDTH) {

inGame = false;

}

if (x[0] < 0) {

inGame = false;

}

if(!inGame) {

timer.stop();

}

}

private void locateApple() {

int r = (int) (Math.random() \* RAND\_POS);

apple\_x = ((r \* DOT\_SIZE));

r = (int) (Math.random() \* RAND\_POS);

apple\_y = ((r \* DOT\_SIZE));

}

@Override

public void actionPerformed(ActionEvent e) {

if (inGame) {

checkApple();

checkCollision();

move();

}

repaint();

}

private class TAdapter extends KeyAdapter {

@Override

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();

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

leftDirection = true;

upDirection = false;

downDirection = false;

}

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

rightDirection = true;

upDirection = false;

downDirection = false;

}

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

upDirection = true;

rightDirection = false;

leftDirection = false;

}

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

downDirection = true;

rightDirection = false;

leftDirection = false;

}

}

}

}

First we will define the constants used in our game.

private final int B\_WIDTH = 300;

private final int B\_HEIGHT = 300;

private final int DOT\_SIZE = 10;

private final int ALL\_DOTS = 900;

private final int RAND\_POS = 29;

private final int DELAY = 140;

The B\_WIDTH and B\_HEIGHT constants determine the size of the board. The DOT\_SIZE is the size of the apple and the dot of the snake. The ALL\_DOTS constant defines the maximum number of possible dots on the board (900 = (300\*300)/(10\*10)). The RAND\_POS constant is used to calculate a random position for an apple. The DELAY constant determines the speed of the game.

private final int x[] = new int[ALL\_DOTS];

private final int y[] = new int[ALL\_DOTS];

These two arrays store the x and y coordinates of all joints of a snake.

private void loadImages() {

ImageIcon iid = new ImageIcon("dot.png");

ball = iid.getImage();

ImageIcon iia = new ImageIcon("apple.png");

apple = iia.getImage();

ImageIcon iih = new ImageIcon("head.png");

head = iih.getImage();

}

In the loadImages() method we get the images for the game. The ImageIcon class is used for displaying PNG images.

private void initGame() {

dots = 3;

for (int z = 0; z < dots; z++) {

x[z] = 50 - z \* 10;

y[z] = 50;

}

locateApple();

timer = new Timer(DELAY, this);

timer.start();

}

In the initGame() method we create the snake, randomly locate an apple on the board, and start the timer.

private void checkApple() {

if ((x[0] == apple\_x) && (y[0] == apple\_y)) {

dots++;

locateApple();

}

}

If the apple collides with the head, we increase the number of joints of the snake. We call thelocateApple() method which randomly positions a new apple object.

In the move() method we have the key algorithm of the game. To understand it, look at how the snake is moving. We control the head of the snake. We can change its direction with the cursor keys. The rest of the joints move one position up the chain. The second joint moves where the first was, the third joint where the second was etc.

for (int z = dots; z > 0; z--) {

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

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

}

This code moves the joints up the chain.

if (leftDirection) {

x[0] -= DOT\_SIZE;

}

This line moves the head to the left.

In the checkCollision() method, we determine if the snake has hit itself or one of the walls.

for (int z = dots; z > 0; z--) {

if ((z > 4) && (x[0] == x[z]) && (y[0] == y[z])) {

inGame = false;

}

}

If the snake hits one of its joints with its head the game is over.

if (y[0] >= B\_HEIGHT) {

inGame = false;

}

The game is finished if the snake hits the bottom of the board.

Snake.java

package com.zetcode;

import java.awt.EventQueue;

import javax.swing.JFrame;

public class Snake extends JFrame {

public Snake() {

add(new Board());

setResizable(false);

pack();

setTitle("Snake");

setLocationRelativeTo(null);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

JFrame ex = new Snake();

ex.setVisible(true);

}

});

}

}

This is the main class.

setResizable(false);

pack();

The setResizable() method affects the insets of the JFrame container on some platforms. Therefore, it is important to call it before the pack() method. Otherwise, the collision of the snake's head with the right and bottom borders might not work correctly.

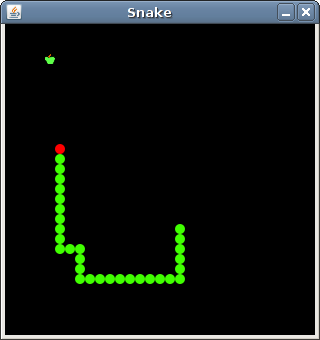


Figure: Snake