**Pacman**

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

Pacman is an arcade game originally developed by a Japanese company *Namco* in 1980. Pacman became one of the most popular arcade games ever created.

**Development**

The following code example is a remake of a Pacman game by Brian Postma available at[http://www.brianpostma.com](http://www.brianpostma.com/java.html). The code is modified and simplified so that it is easier to understand.

The goal of the game is to collect all the points in the maze and avoid the ghosts. The Pacman is animated in two ways: his position in the maze and his body. We animate his body with four images, depending on the direction. The animation is used to create the illusion of Pacman opening and closing his mouth. The maze consists of 15x15 squares. The structure of the maze is based on a simple array of integers. Pacman has three lives. We also count the score.

The game consists of two files: Board.java and Pacman.java.

Board.java

package com.zetcode;

import java.awt.BasicStroke;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Event;

import java.awt.Font;

import java.awt.FontMetrics;

import java.awt.Graphics;

import java.awt.Graphics2D;

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 Dimension d;

private final Font smallfont = new Font("Helvetica", Font.BOLD, 14);

private Image ii;

private final Color dotcolor = new Color(192, 192, 0);

private Color mazecolor;

private boolean ingame = false;

private boolean dying = false;

private final int blocksize = 24;

private final int nrofblocks = 15;

private final int scrsize = nrofblocks \* blocksize;

private final int pacanimdelay = 2;

private final int pacmananimcount = 4;

private final int maxghosts = 12;

private final int pacmanspeed = 6;

private int pacanimcount = pacanimdelay;

private int pacanimdir = 1;

private int pacmananimpos = 0;

private int nrofghosts = 6;

private int pacsleft, score;

private int[] dx, dy;

private int[] ghostx, ghosty, ghostdx, ghostdy, ghostspeed;

private Image ghost;

private Image pacman1, pacman2up, pacman2left, pacman2right, pacman2down;

private Image pacman3up, pacman3down, pacman3left, pacman3right;

private Image pacman4up, pacman4down, pacman4left, pacman4right;

private int pacmanx, pacmany, pacmandx, pacmandy;

private int reqdx, reqdy, viewdx, viewdy;

private final short leveldata[] = {

19, 26, 26, 26, 18, 18, 18, 18, 18, 18, 18, 18, 18, 18, 22,

21, 0, 0, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,

21, 0, 0, 0, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20,

21, 0, 0, 0, 17, 16, 16, 24, 16, 16, 16, 16, 16, 16, 20,

17, 18, 18, 18, 16, 16, 20, 0, 17, 16, 16, 16, 16, 16, 20,

17, 16, 16, 16, 16, 16, 20, 0, 17, 16, 16, 16, 16, 24, 20,

25, 16, 16, 16, 24, 24, 28, 0, 25, 24, 24, 16, 20, 0, 21,

1, 17, 16, 20, 0, 0, 0, 0, 0, 0, 0, 17, 20, 0, 21,

1, 17, 16, 16, 18, 18, 22, 0, 19, 18, 18, 16, 20, 0, 21,

1, 17, 16, 16, 16, 16, 20, 0, 17, 16, 16, 16, 20, 0, 21,

1, 17, 16, 16, 16, 16, 20, 0, 17, 16, 16, 16, 20, 0, 21,

1, 17, 16, 16, 16, 16, 16, 18, 16, 16, 16, 16, 20, 0, 21,

1, 17, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 20, 0, 21,

1, 25, 24, 24, 24, 24, 24, 24, 24, 24, 16, 16, 16, 18, 20,

9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 25, 24, 24, 24, 28

};

private final int validspeeds[] = {1, 2, 3, 4, 6, 8};

private final int maxspeed = 6;

private int currentspeed = 3;

private short[] screendata;

private Timer timer;

public Board() {

loadImages();

initVariables();

addKeyListener(new TAdapter());

setFocusable(true);

setBackground(Color.black);

setDoubleBuffered(true);

}

private void initVariables() {

screendata = new short[nrofblocks \* nrofblocks];

mazecolor = new Color(5, 100, 5);

d = new Dimension(400, 400);

ghostx = new int[maxghosts];

ghostdx = new int[maxghosts];

ghosty = new int[maxghosts];

ghostdy = new int[maxghosts];

ghostspeed = new int[maxghosts];

dx = new int[4];

dy = new int[4];

timer = new Timer(40, this);

timer.start();

}

@Override

public void addNotify() {

super.addNotify();

initGame();

}

private void doAnim() {

pacanimcount--;

if (pacanimcount <= 0) {

pacanimcount = pacanimdelay;

pacmananimpos = pacmananimpos + pacanimdir;

if (pacmananimpos == (pacmananimcount - 1) || pacmananimpos == 0) {

pacanimdir = -pacanimdir;

}

}

}

private void playGame(Graphics2D g2d) {

if (dying) {

death();

} else {

movePacman();

drawPacman(g2d);

moveGhosts(g2d);

checkMaze();

}

}

private void showIntroScreen(Graphics2D g2d) {

g2d.setColor(new Color(0, 32, 48));

g2d.fillRect(50, scrsize / 2 - 30, scrsize - 100, 50);

g2d.setColor(Color.white);

g2d.drawRect(50, scrsize / 2 - 30, scrsize - 100, 50);

String s = "Press s to start.";

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

FontMetrics metr = this.getFontMetrics(small);

g2d.setColor(Color.white);

g2d.setFont(small);

g2d.drawString(s, (scrsize - metr.stringWidth(s)) / 2, scrsize / 2);

}

private void drawScore(Graphics2D g) {

int i;

String s;

g.setFont(smallfont);

g.setColor(new Color(96, 128, 255));

s = "Score: " + score;

g.drawString(s, scrsize / 2 + 96, scrsize + 16);

for (i = 0; i < pacsleft; i++) {

g.drawImage(pacman3left, i \* 28 + 8, scrsize + 1, this);

}

}

private void checkMaze() {

short i = 0;

boolean finished = true;

while (i < nrofblocks \* nrofblocks && finished) {

if ((screendata[i] & 48) != 0) {

finished = false;

}

i++;

}

if (finished) {

score += 50;

if (nrofghosts < maxghosts) {

nrofghosts++;

}

if (currentspeed < maxspeed) {

currentspeed++;

}

initLevel();

}

}

private void death() {

pacsleft--;

if (pacsleft == 0) {

ingame = false;

}

continueLevel();

}

private void moveGhosts(Graphics2D g2d) {

short i;

int pos;

int count;

for (i = 0; i < nrofghosts; i++) {

if (ghostx[i] % blocksize == 0 && ghosty[i] % blocksize == 0) {

pos = ghostx[i] / blocksize + nrofblocks \* (int) (ghosty[i] / blocksize);

count = 0;

if ((screendata[pos] & 1) == 0 && ghostdx[i] != 1) {

dx[count] = -1;

dy[count] = 0;

count++;

}

if ((screendata[pos] & 2) == 0 && ghostdy[i] != 1) {

dx[count] = 0;

dy[count] = -1;

count++;

}

if ((screendata[pos] & 4) == 0 && ghostdx[i] != -1) {

dx[count] = 1;

dy[count] = 0;

count++;

}

if ((screendata[pos] & 8) == 0 && ghostdy[i] != -1) {

dx[count] = 0;

dy[count] = 1;

count++;

}

if (count == 0) {

if ((screendata[pos] & 15) == 15) {

ghostdx[i] = 0;

ghostdy[i] = 0;

} else {

ghostdx[i] = -ghostdx[i];

ghostdy[i] = -ghostdy[i];

}

} else {

count = (int) (Math.random() \* count);

if (count > 3) {

count = 3;

}

ghostdx[i] = dx[count];

ghostdy[i] = dy[count];

}

}

ghostx[i] = ghostx[i] + (ghostdx[i] \* ghostspeed[i]);

ghosty[i] = ghosty[i] + (ghostdy[i] \* ghostspeed[i]);

drawGhost(g2d, ghostx[i] + 1, ghosty[i] + 1);

if (pacmanx > (ghostx[i] - 12) && pacmanx < (ghostx[i] + 12)

&& pacmany > (ghosty[i] - 12) && pacmany < (ghosty[i] + 12)

&& ingame) {

dying = true;

}

}

}

private void drawGhost(Graphics2D g2d, int x, int y) {

g2d.drawImage(ghost, x, y, this);

}

private void movePacman() {

int pos;

short ch;

if (reqdx == -pacmandx && reqdy == -pacmandy) {

pacmandx = reqdx;

pacmandy = reqdy;

viewdx = pacmandx;

viewdy = pacmandy;

}

if (pacmanx % blocksize == 0 && pacmany % blocksize == 0) {

pos = pacmanx / blocksize + nrofblocks \* (int) (pacmany / blocksize);

ch = screendata[pos];

if ((ch & 16) != 0) {

screendata[pos] = (short) (ch & 15);

score++;

}

if (reqdx != 0 || reqdy != 0) {

if (!((reqdx == -1 && reqdy == 0 && (ch & 1) != 0)

|| (reqdx == 1 && reqdy == 0 && (ch & 4) != 0)

|| (reqdx == 0 && reqdy == -1 && (ch & 2) != 0)

|| (reqdx == 0 && reqdy == 1 && (ch & 8) != 0))) {

pacmandx = reqdx;

pacmandy = reqdy;

viewdx = pacmandx;

viewdy = pacmandy;

}

}

// Check for standstill

if ((pacmandx == -1 && pacmandy == 0 && (ch & 1) != 0)

|| (pacmandx == 1 && pacmandy == 0 && (ch & 4) != 0)

|| (pacmandx == 0 && pacmandy == -1 && (ch & 2) != 0)

|| (pacmandx == 0 && pacmandy == 1 && (ch & 8) != 0)) {

pacmandx = 0;

pacmandy = 0;

}

}

pacmanx = pacmanx + pacmanspeed \* pacmandx;

pacmany = pacmany + pacmanspeed \* pacmandy;

}

private void drawPacman(Graphics2D g2d) {

if (viewdx == -1) {

drawPacnanLeft(g2d);

} else if (viewdx == 1) {

drawPacmanRight(g2d);

} else if (viewdy == -1) {

drawPacmanUp(g2d);

} else {

drawPacmanDown(g2d);

}

}

private void drawPacmanUp(Graphics2D g2d) {

switch (pacmananimpos) {

case 1:

g2d.drawImage(pacman2up, pacmanx + 1, pacmany + 1, this);

break;

case 2:

g2d.drawImage(pacman3up, pacmanx + 1, pacmany + 1, this);

break;

case 3:

g2d.drawImage(pacman4up, pacmanx + 1, pacmany + 1, this);

break;

default:

g2d.drawImage(pacman1, pacmanx + 1, pacmany + 1, this);

break;

}

}

private void drawPacmanDown(Graphics2D g2d) {

switch (pacmananimpos) {

case 1:

g2d.drawImage(pacman2down, pacmanx + 1, pacmany + 1, this);

break;

case 2:

g2d.drawImage(pacman3down, pacmanx + 1, pacmany + 1, this);

break;

case 3:

g2d.drawImage(pacman4down, pacmanx + 1, pacmany + 1, this);

break;

default:

g2d.drawImage(pacman1, pacmanx + 1, pacmany + 1, this);

break;

}

}

private void drawPacnanLeft(Graphics2D g2d) {

switch (pacmananimpos) {

case 1:

g2d.drawImage(pacman2left, pacmanx + 1, pacmany + 1, this);

break;

case 2:

g2d.drawImage(pacman3left, pacmanx + 1, pacmany + 1, this);

break;

case 3:

g2d.drawImage(pacman4left, pacmanx + 1, pacmany + 1, this);

break;

default:

g2d.drawImage(pacman1, pacmanx + 1, pacmany + 1, this);

break;

}

}

private void drawPacmanRight(Graphics2D g2d) {

switch (pacmananimpos) {

case 1:

g2d.drawImage(pacman2right, pacmanx + 1, pacmany + 1, this);

break;

case 2:

g2d.drawImage(pacman3right, pacmanx + 1, pacmany + 1, this);

break;

case 3:

g2d.drawImage(pacman4right, pacmanx + 1, pacmany + 1, this);

break;

default:

g2d.drawImage(pacman1, pacmanx + 1, pacmany + 1, this);

break;

}

}

private void drawMaze(Graphics2D g2d) {

short i = 0;

int x, y;

for (y = 0; y < scrsize; y += blocksize) {

for (x = 0; x < scrsize; x += blocksize) {

g2d.setColor(mazecolor);

g2d.setStroke(new BasicStroke(2));

if ((screendata[i] & 1) != 0) {

g2d.drawLine(x, y, x, y + blocksize - 1);

}

if ((screendata[i] & 2) != 0) {

g2d.drawLine(x, y, x + blocksize - 1, y);

}

if ((screendata[i] & 4) != 0) {

g2d.drawLine(x + blocksize - 1, y, x + blocksize - 1,

y + blocksize - 1);

}

if ((screendata[i] & 8) != 0) {

g2d.drawLine(x, y + blocksize - 1, x + blocksize - 1,

y + blocksize - 1);

}

if ((screendata[i] & 16) != 0) {

g2d.setColor(dotcolor);

g2d.fillRect(x + 11, y + 11, 2, 2);

}

i++;

}

}

}

private void initGame() {

pacsleft = 3;

score = 0;

initLevel();

nrofghosts = 6;

currentspeed = 3;

}

private void initLevel() {

int i;

for (i = 0; i < nrofblocks \* nrofblocks; i++) {

screendata[i] = leveldata[i];

}

continueLevel();

}

private void continueLevel() {

short i;

int dx = 1;

int random;

for (i = 0; i < nrofghosts; i++) {

ghosty[i] = 4 \* blocksize;

ghostx[i] = 4 \* blocksize;

ghostdy[i] = 0;

ghostdx[i] = dx;

dx = -dx;

random = (int) (Math.random() \* (currentspeed + 1));

if (random > currentspeed) {

random = currentspeed;

}

ghostspeed[i] = validspeeds[random];

}

pacmanx = 7 \* blocksize;

pacmany = 11 \* blocksize;

pacmandx = 0;

pacmandy = 0;

reqdx = 0;

reqdy = 0;

viewdx = -1;

viewdy = 0;

dying = false;

}

private void loadImages() {

ghost = new ImageIcon("images/ghost.png").getImage();

pacman1 = new ImageIcon("images/pacman.png").getImage();

pacman2up = new ImageIcon("images/up1.png").getImage();

pacman3up = new ImageIcon("images/up2.png").getImage();

pacman4up = new ImageIcon("images/up3.png").getImage();

pacman2down = new ImageIcon("images/down1.png").getImage();

pacman3down = new ImageIcon("images/down2.png").getImage();

pacman4down = new ImageIcon("images/down3.png").getImage();

pacman2left = new ImageIcon("images/left1.png").getImage();

pacman3left = new ImageIcon("images/left2.png").getImage();

pacman4left = new ImageIcon("images/left3.png").getImage();

pacman2right = new ImageIcon("images/right1.png").getImage();

pacman3right = new ImageIcon("images/right2.png").getImage();

pacman4right = new ImageIcon("images/right3.png").getImage();

}

@Override

public void paintComponent(Graphics g) {

super.paintComponent(g);

doDrawing(g);

}

private void doDrawing(Graphics g) {

Graphics2D g2d = (Graphics2D) g;

g2d.setColor(Color.black);

g2d.fillRect(0, 0, d.width, d.height);

drawMaze(g2d);

drawScore(g2d);

doAnim();

if (ingame) {

playGame(g2d);

} else {

showIntroScreen(g2d);

}

g2d.drawImage(ii, 5, 5, this);

Toolkit.getDefaultToolkit().sync();

g2d.dispose();

}

class TAdapter extends KeyAdapter {

@Override

public void keyPressed(KeyEvent e) {

int key = e.getKeyCode();

if (ingame) {

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

reqdx = -1;

reqdy = 0;

} else if (key == KeyEvent.VK\_RIGHT) {

reqdx = 1;

reqdy = 0;

} else if (key == KeyEvent.VK\_UP) {

reqdx = 0;

reqdy = -1;

} else if (key == KeyEvent.VK\_DOWN) {

reqdx = 0;

reqdy = 1;

} else if (key == KeyEvent.VK\_ESCAPE && timer.isRunning()) {

ingame = false;

} else if (key == KeyEvent.VK\_PAUSE) {

if (timer.isRunning()) {

timer.stop();

} else {

timer.start();

}

}

} else {

if (key == 's' || key == 'S') {

ingame = true;

initGame();

}

}

}

@Override

public void keyReleased(KeyEvent e) {

int key = e.getKeyCode();

if (key == Event.LEFT || key == Event.RIGHT

|| key == Event.UP || key == Event.DOWN) {

reqdx = 0;

reqdy = 0;

}

}

}

@Override

public void actionPerformed(ActionEvent e) {

repaint();

}

}

The Pacman is controlled with the cursor keys. The Esc key finishes the game, the Pause key pauses it.

private int pacmanx, pacmany, pacmandx, pacmandy;

The first two variables store the x and y coordinates of the Pacman sprite. The last two variables are the delta changes in horizontal and vertical directions.

private final short leveldata[] = {

19, 26, 26, 26, 18, 18, 18, 18, ...

};

These numbers make up the maze. They provide information out of which we create the corners and the points. Number 1 is a left corner. Numbers 2, 4 and 8 represent top, right, bottom corners respectively. Number 16 is a point. These number can be added, for example number 19 in the upper left corner means that the square will have top and left borders and a point (16 + 2 + 1).

private void doAnim() {

pacanimcount--;

if (pacanimcount <= 0) {

pacanimcount = pacanimdelay;

pacmananimpos = pacmananimpos + pacanimdir;

if (pacmananimpos == (pacmananimcount - 1) || pacmananimpos == 0) {

pacanimdir = -pacanimdir;

}

}

}

The doAnim() counts the pacmananimpos variable which determines what pacman image is drawn. There are four pacman images. There is also a pacanimdelay variable which makes the animation a bit slower. Otherwise the pacman would open his mouth too fast.

boolean finished = true;

while (i < nrofblocks \* nrofblocks && finished) {

if ((screendata[i] & 48) != 0) {

finished = false;

}

i++;

}

This code is part of the checkMaze() method. It checks if there are any points left for the Pacman to eat. Number 16 stands for a point. If all points are consumed, we move to the next level. (In our case, we just restart the game.)

Next we will examine the moveGhosts() method. The ghosts move one square and then decide if they change the direction.

if (ghostx[i] % blocksize == 0 && ghosty[i] % blocksize == 0) {

We continue only if we have finished moving one square.

pos = ghostx[i] / blocksize + nrofblocks \* (int)(ghosty[i] / blocksize);

This line determines where the ghost is situated. In which position/square. There are 225 theoretical positions. (A ghost cannot move over walls.)

if ((screendata[pos] & 1) == 0 && ghostdx[i] != 1) {

dx[count] = -1;

dy[count] = 0;

count++;

}

If there is no obstacle on the left and the ghost is not already moving to the right, the ghost will move to the left. What does this code really mean? If the ghost enters a tunnel, he will continue in the same direction until he is out of the tunnel. Moving of ghosts is partly random. We do not apply this randomness inside long tunnels. The ghost might get stuck there.

if (pacmanx > (ghostx[i] - 12) && pacmanx < (ghostx[i] + 12)

&& pacmany > (ghosty[i] - 12) && pacmany < (ghosty[i] + 12)

&& ingame) {

dying = true;

}

If there is a collision between ghosts and a Pacman, the Pacman dies.

Next we are going to examine the movePacman() method. The reqdx and reqdy variables are determined in the TAdapter inner class. These variables are controlled with cursor keys.

if ((ch & 16) != 0) {

screendata[pos] = (short) (ch & 15);

score++;

}

If the pacman moves to a position with a point, we remove it from the maze and increase the score value.

if ((pacmandx == -1 && pacmandy == 0 && (ch & 1) != 0) ||

(pacmandx == 1 && pacmandy == 0 && (ch & 4) != 0) ||

(pacmandx == 0 && pacmandy == -1 && (ch & 2) != 0) ||

(pacmandx == 0 && pacmandy == 1 && (ch & 8) != 0)) {

pacmandx = 0;

pacmandy = 0;

}

The Pacman stops if he cannot move further it his current direction.

private void drawPacman(Graphics2D g2d) {

if (viewdx == -1) {

drawPacnanLeft(g2d);

} else if (viewdx == 1) {

drawPacmanRight(g2d);

} else if (viewdy == -1) {

drawPacmanUp(g2d);

} else {

drawPacmanDown(g2d);

}

}

There are four possible directions for a Pacman. There are four images for all directions. The images are used to animate Pacman opening and closing his mouth.

The drawMaze() method draws the maze out of the numbers in the screendata array. Number 1 is a left border, 2 is a top border, 4 is a right border, 8 is a bottom border and 16 is a point. We simply go through all 225 squares in the maze. For example we have 9 in the screendata array. We have the first bit (1) and the fourth bit (8) set. So we draw a bottom and a left border on this particular square.

if ((screendata[i] & 1) != 0) {

g2d.drawLine(x, y, x, y + blocksize - 1);

}

We draw a left border if the first bit of a number is set.

Pacman.java

package com.zetcode;

import java.awt.EventQueue;

import javax.swing.JFrame;

public class Pacman extends JFrame {

public Pacman() {

initUI();

}

private void initUI() {

add(new Board());

setTitle("Pacman");

setDefaultCloseOperation(EXIT\_ON\_CLOSE);

setSize(380, 420);

setLocationRelativeTo(null);

setVisible(true);

}

public static void main(String[] args) {

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

Pacman ex = new Pacman();

ex.setVisible(true);

}

});

}

}

This is a Pacman file with a main method.

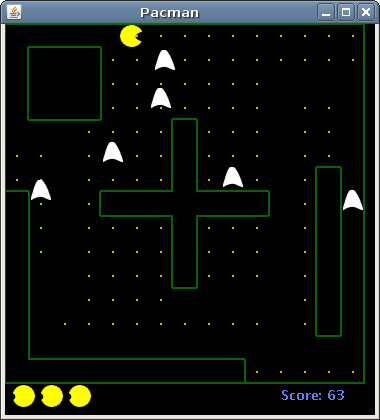


Figure: Pacman