**Практическая работа 4.5. Реализация третьего уровня создания игры змейка.**

Полный вариант игры строится на базе второго уровня сложности,поэтому мы продолжаем дорабатывать существующий проект.

Вместе с техническим заданием по разработке игры Змейка был предложен алгоритм реализации этой игры. Речь идет об алгоритме перемещения тела змейки по игровому полю. Все пустые ячейки игрового поля в двумерном массиве отмечены нулями(см. рис. 3),объект для поедания отмечен минус единицей: -1 (см. рис. 3), голова змейки отмечена значением единица: 1 (см. рис. 3), тело змейки будет помечено числами от 2 и более (см. рис. 3).

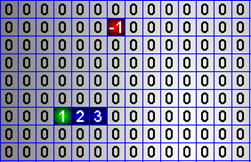


Рис. 3

В третьем уровне сложности нам предстоит добавить тело змейки. После поедания объекта длина змейки увеличивается на единицу, увеличение происходит с конца. Для рассмотренного на рисунке примера к змейке из трех клеток добавится четвертая и будет отмечена числом: 4 (см. рис.4)

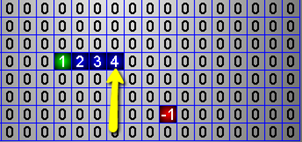


Рис. 4

При следующем поедании объекта к змейке из четырех клеток добавится пятая и будет отмечена числом:5 (см. рис. 5).Таким образом, в хвосте змейки всегда будет стоять число, равное длине змейки. Если длина змейки пять, то и в хвосте будет пять клеток (см. рис. 5). Почему мы не сделали проще, не отметили все тело змейки числом, например два?

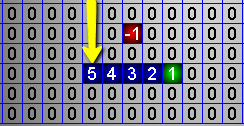


Рис. 5

Дело в том, что когда змейка будет сильно изогнута, и части ее тела будут прилегать друг к другу, то клетки ее тела будут как бы перемешиваться,и какая из них будет являться хвостом неизвестно (см. рис. 6).

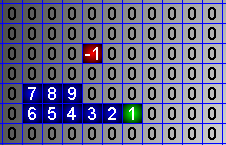


Рис. 6

Безусловно, данный алгоритм не является единственным! Можно придумать другие варианты!

Начнем с доработки файла zmeika.java,дополним программный код в классе myPanel. Найдем место, где у нас реализован таймер для изменения логики игры –tmUpdate (см. рис. 7)

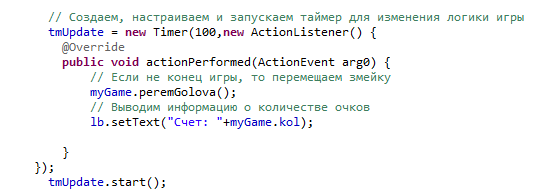


Рис. 7

Нам нужно, чтобы перемещалась не только голова змейки, но и ее тело. Для этого в классе game мы добавим метод perem(), который будет отвечать за перемещение змейки. Поэтому удалим вызов метода peremGolova(), а вместо него добавим следующий программный код:

//Создаем, настраиваем и запускаем таймер для

//изменения логики игры

tmUpdate = new Timer(100,new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

// Если не конец игры, то перемещаем змейку

if (myGame.endg==false)

{

myGame.perem();

}

// Выводим информацию о количестве очков

lb.setText("Счет: "+myGame.kol);

}

});

tmUpdate.start();

Переменная endg также будет добавлена в классе game. Эта переменная логического типа будет хранить признак "конца игры". В конце игры ее значение будет: true. Другими словами, мы будем вызывать метод перемещения змейки пока еще не конец игры. Метод perem() будет изменять массив. Следующие доработки коснутся изменения направления движения змейки при нажатии на стрелки клавиатуры. Вместо изменения переменной napr мы будем изменять переменную new\_napr, которую добавим в классе game:

// Если нажатие одной из четырех стрелочек,то

// изменение направления змейки

if (key==KeyEvent.VK\_LEFT) myGame.new\_napr = 0;

else if (key==KeyEvent.VK\_UP) myGame.new\_napr = 1;

else if (key==KeyEvent.VK\_RIGHT) myGame.new\_napr = 2;

else if (key==KeyEvent.VK\_DOWN) myGame.new\_napr = 3;

Теперь перейдем к методу paintComponent(). В самом конце метода добавим вывод изображения окончания игры:

//Вывод изображения конца игры - при окончании игры

if (myGame.endg==true)

{

gr.drawImage(endg,250,200,200,100,null);

}

Если признак окончания игры равен true,то будет выводиться картинка, загруженная в переменную endg. Так как строки, описывающие этот алгоритм, будут находиться в самом конце метода paintComponent(), то изображение, обозначающее конец игры, будет выводиться самым последним, и будет находиться поверх остальных изображений (см. рис. 8):



Рис. 8

Теперь в методе paintComponent() нам нужно найти два вложенных цикла для отрисовки змейки(см. рис. 9). Сейчас этот цикл рисует голову змейки и объект для поедания. Добавим фрагмент кода, который будет прорисовывать тело змейки.

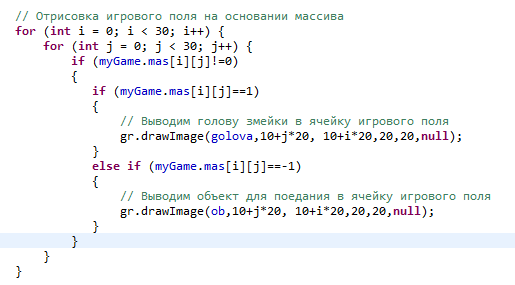


Рис. 9

// Отрисовка игрового поля на основании массива

for (int i = 0; i < 30; i++) {

for (int j = 0; j < 30; j++) {

if (myGame.mas[i][j]!=0)

{

if (myGame.mas[i][j]==1)

{

// Выводим голову змейки в ячейку игрового поля

gr.drawImage(golova,10+j\*20, 10+i\*20,20,20,null);

}

else if (myGame.mas[i][j]==-1)

{

// Выводим объект для поедания в ячейку игрового поля

gr.drawImage(ob,10+j\*20, 10+i\*20,20,20,null);

}

else if (myGame.mas[i][j]>=2)

{

// Выводим тело змейки

gr.drawImage(telo,10+j\*20, 10+i\*20,20,20,null);

}

}

}

}

Мы добавили блок:

else if (myGame.mas[i][j]>=2)

{

// Выводим тело змейки

gr.drawImage(telo,10+j\*20, 10+i\*20,20,20,null);

}

Если значение элемента массива больше или равно двум, то выводится изображение из переменной telo, в которую мы загрузили картинку элемента тела змейки.Как было рассмотрено выше –тело змейки обозначено числами от двух и больше. На этом доработки в классе myPanel и файле zmeika.java для третьего уровня сложности закончены. Полный программный код файла zmeika.javaвыглядит так:

// Для обработки событий

import java.awt.event.\*;

// Для работы с окнами

import javax.swing.\*;

// Для работы с графикой

import java.awt.\*;

// Для работы с изображениями

import javax.imageio.\*;

// Для работы с файлами

import java.io.\*;

// Главный класс программы

public class zmeika

{

// Метод запуска приложения

public static void main(String[] args)

{

//Создание объекта окна игрового поля

myFrame okno = new myFrame();

}

}

// Класс окна игрового поля

class myFrame extends JFrame

{

// Конструктор класса

public myFrame()

{

//Создание объекта панели и подключения ее к окну

myPanel pan = new myPanel();

Container cont = getContentPane();

cont.add(pan);

//Заголовок окна

setTitle("Игра \"Змейка\"");

//Границы окна: расположение и размеры

setBounds(0, 0, 800, 650);

//Операция при закрытии окна - завершение приложения

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

//Запрет изменения размеров окна

setResizable(false);

//Отображение (показ) окна

setVisible(true);

}

}

// Класс панели игрового поля

class myPanel extends JPanel

{

// Переменная для реализации логики игры

private game myGame;

// Два таймера: отрисовки и изменения логики игры

private Timer tmDraw, tmUpdate;

// Изображения, используемые в игре

private Image fon,telo,golova,ob,endg;

// Надпись для количества очков

private JLabel lb;

// Две кнопки

private JButton btn1,btn2;

// Ссылка на панель

private myPanel pan;

// Класс для обработки событий от клавиатуры

private class myKey implements KeyListener

{

//Метод при нажатии на клавишу

public void keyPressed(KeyEvent e)

{

// Получение кода нажатой клавиши

int key = e.getKeyCode();

// Если нажатие одной из четырех стрелочек, то

// изменение направления змейки

if (key==KeyEvent.VK\_LEFT) myGame. new\_napr = 0;

else if (key==KeyEvent.VK\_UP) myGame. new\_napr = 1;

else if (key==KeyEvent.VK\_RIGHT) myGame. new\_napr = 2;

else if (key==KeyEvent.VK\_DOWN) myGame. new\_napr = 3;

}

public void keyReleased(KeyEvent e) {}

public void keyTyped(KeyEvent e) {}

}

// Конструктор класса

public myPanel()

{

//Помещаем ссылку на саму панель в переменную

pan = this;

//Подключение обработчика события для клавиатуры к панели

this.addKeyListener(new myKey());

// Делаем панель в фокусе - для приема событий от клавиатуры

this.setFocusable(true);

//Попытка загрузки всех изображений для игры

try

{

fon = ImageIO.read(new File("c:\\fon.png"));

telo = ImageIO.read(new File("c:\\telo.png"));

golova = ImageIO.read(new File("c:\\golova.png"));

ob = ImageIO.read(new File("c:\\ob.png"));

endg = ImageIO.read(new File("c:\\endg.png"));

}

catch (Exception ex) {}

//Создаем объект новой игры

myGame = new game();

myGame.start();

//Создаем, настраиваем и запускаем таймер

// для отрисовки игрового поля

tmDraw = new Timer(20,new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

// Вызываем перерисовку -paintComponent()

repaint();

}

});

tmDraw.start();

//Создаем, настраиваем и запускаем таймер

// для изменения логики игры

tmUpdate = new Timer(100,new ActionListener() {

@Override

public void actionPerformed(ActionEvent arg0) {

// Если не конец игры, то перемещаем змейку

if (myGame.endg==false)

{

myGame.perem();

}

// Выводим информацию о количестве очков

lb.setText("Счет: "+myGame.kol);

}

});

tmUpdate.start();

//Включаем возможность произвольного размещения

//элементов интерфейса на панели

setLayout(null);

//Создаем текстовую надпись

lb = new JLabel("Счет: 0");

lb.setForeground(Color.WHITE);

lb.setFont(new Font("serif",0,30));

lb.setBounds(630, 200, 150, 50);

add(lb);

//Создаем кнопку Новая игра

btn1 = new JButton();

btn1.setText("Новая игра");

btn1.setForeground(Color.BLUE);

btn1.setFont(new Font("serif",0,20));

btn1.setBounds(630, 30, 150, 50);

btn1.addActionListener(new ActionListener() {

// Обработчик события при нажатии на кнопку Новая игра

public void actionPerformed(ActionEvent arg0) {

// Запуск игры

myGame.start();

// Забираем фокус у кнопки Новая игры

btn1.setFocusable(false);

// Забираем фокус у кнопки Выход

btn2.setFocusable(false);

// Отдаем фокус панели

pan.setFocusable(true);

}

});

add(btn1);

//Создаем кнопку Выход

btn2 = new JButton();

btn2.setText("Выход");

btn2.setForeground(Color.RED);

btn2.setFont(new Font("serif",0,20));

btn2.setBounds(630, 100, 150, 50);

btn2.addActionListener(new ActionListener() {

// Обработчик события при нажатии на кнопку Новая игра

public void actionPerformed(ActionEvent arg0) {

// Выход их игры -завершение работы приложения

System.exit(0);

}

});

add(btn2);

}

// Метод отрисовки

public void paintComponent(Graphics gr)

{

//Очищение игрового поля

super.paintComponent(gr);

//Отрисовка фона

gr.drawImage(fon,0,0,800,650,null);

//Отрисовка игрового поля на основании массива

for (int i = 0; i < 30; i++) {

for (int j = 0; j < 30; j++) {

if (myGame.mas[i][j]!=0)

{

if (myGame.mas[i][j]==1)

{

// Выводим голову змейки в ячейку игрового поля

gr.drawImage(golova,10+j\*20, 10+i\*20,20,20,null);

}

else if (myGame.mas[i][j]==-1)

{

// Выводим объект для поедания в ячейку игрового поля

gr.drawImage(ob,10+j\*20, 10+i\*20,20,20,null);

}

else if (myGame.mas[i][j]>=2)

{

// Выводим тело змейки

gr.drawImage(telo,10+j\*20, 10+i\*20,20,20,null);

}

}

}

}

//Отрисовка сетки игрового поля из синих линий

gr.setColor(Color.BLUE);

for (int i = 0; i <= 30; i++)

{

// Рисование линий сетки

gr.drawLine(10+i\*20, 10, 10+i\*20, 610);

gr.drawLine(10, 10+i\*20, 610, 10+i\*20);

}

//Вывод изображения конца игры - при окончании игры

if (myGame.endg==true)

{

gr.drawImage(endg,250,200,200,100,null);

}

}

}

Теперь перейдем в файл game.java и приступим к доработкам класса game. В первую очередь добавим свойства (переменные) класса,которые нам понадобятся.

public class game

{

// Двухмерный массив для хранения игрового поля

public int[][] mas;

// Текущее направление движения

public int napr;

// Координаты головы змейки

private int gX, gY;

// Количество очков

public int kol;

// Новое направление движения

public int new\_napr;

// Длина змейки

private int dlina;

// Признак окончания игры

public boolean endg;

Мы добавили три новые переменные.

1. Переменная new\_napr–это направление движения змейки, которое выбирает пользователь.

Когда змейка будет иметь не только голову, но и тело, мы заблокируем возможность изменять направление на противоположное. А в случае,когда направление выбирается тоже самое–нет смысла изменять его вообще. Другими словами,направление будет меняться, если пользователь выбирает движение влево или вправо от текущего. Сначала мы будем проверять выбор пользователя по переменной new\_napr, и если оно нам подходит, то будем изменять значение переменной napr.

2. Переменная dlina будет хранить текущую длину змейки.

3. Переменная endg –это признак окончания игры,переменная логического типа.

После блока объявления переменных добавим новый метод:

// Изменение направления движения змейки (поворот)

private void povorot()

{

if (Math.abs((new\_napr-napr))!=2)

{

napr=new\_napr;

}

}

Метод povorot()будет анализировать выбираемое направление пользователем, хранящееся в переменной new\_napr, и, если возможно изменить направление, переменная napr будет получать новое значение.

Math –это класс математических операций, метод abs() –вычисляет модуль числа –отбрасывает знак минус, если он есть. Если разница между текущим и выбираемым направлением не равна двум, то мы изменяем значение переменной napr на новое.

Теперь перейдем в метод start() изменим две строки(см. рис. 10):

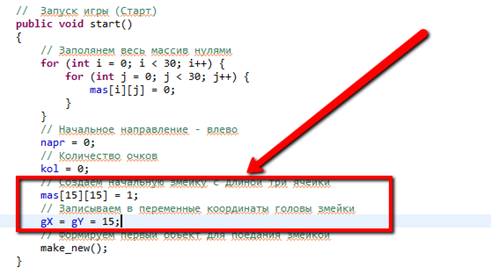


Рис. 10

Вместо двух строк,выделенных на рис. 10, добавим следующие:

//Создаем начальную змейку с длиной три ячейки

mas[14][14] = 1;

mas[14][15] = 2;

mas[14][16] = 3;

dlina = 3;

//Записываем в переменные координаты головы змейки

gX = gY = 14;

//Признак окончания игры

endg=false;

Начальная змейка состоит из трех клеток. Поэтому вписываем в массив три цифры от 1 до 3.

//Создаем начальную змейку с длиной три ячейки

mas[14][14] = 1;

mas[14][15] = 2;

mas[14][16] = 3;

При такой записи змейка будет горизонтальной и расположенной по центру игрового поля.

Укажем длину змейки:

dlina = 3;

В переменные gX и gY записываем число 14. Это координаты головы змейки.

// Признак окончания игры

endg=false;

Признак окончания игры устанавливаем в значение false, т.к. игра только начинается.

Перейдем к методу peremGolova(), этот метод мы несколько изменим. После доработки этот метод будет возвращать некоторое значение –число целого типа. Укажем тип возвращаемого значения:

// Перемещение головы змейки

public int peremGolova()

{

После перемещения головы змейки –метод будет возвращать тот или иной результат перемещения в зависимости от условия:

· если съеден объект, то 1

· если перемещение в пустую ячейку, то2

· если перемещение в тело змейки, то3

Далее по этому значению мы будем определять дальнейшие действия.

Удалим первую строку метода peremGolova() (см. рис.11)

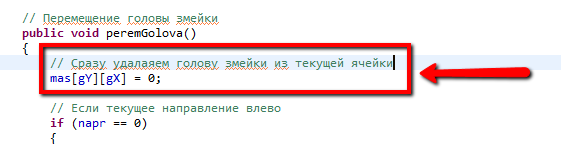


Рис. 11

Удалять голову змейки из текущей ячейки игрового поля мы сразу не будем - блок анализа выхода за пределы экрана останется без изменений. А в конце метода peremGolova() вместо имеющегося программного кода(см. рис. 12) добавим следующий:

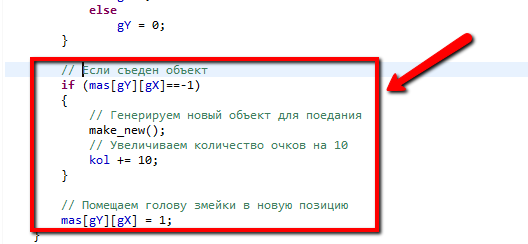


Рис. 12

//Результат

int rez=0;

// Если съеден объект

if (mas[gY][gX]==-1) rez=1;

// Если перемещение в пустую ячейку

else if (mas[gY][gX]==0) rez=2;

// Если попадание в туловище змейки

else if (mas[gY][gX]>0) rez=3;

// В место перемещения головы

//установим значение - минус два

mas[gY][gX] = -2;

//Возвращаем результат

return rez;

В переменной rez будет находиться результат, возвращаемый методом. В зависимости от ситуации он будет равен 1 или 2 или 3. В новую позицию для перемещения головы помещаем значение минус два:

//установим значение - минус два

mas[gY][gX] = -2;

И возвращаем результат:

//Возвращаем результат

return rez;

В новом виде метод peremGolova() будет выглядеть так:

// Перемещение головы змейки

public int peremGolova()

{

// Если текущее направление влево

if (napr == 0)

{

if ((gX - 1) >= 0)

gX--;

else

gX = 29;

}

// Если текущее направление вверх

else if (napr == 1)

{

if ((gY - 1) >= 0)

gY--;

else

gY = 29;

}

// Если текущее направление вправо

else if (napr == 2)

{

if ((gX + 1) <= 29)

gX++;

else

gX = 0;

}

// Если текущее направление вниз

else if (napr == 3)

{

if ((gY + 1) <= 29)

gY++;

else

gY = 0;

}

//Результат

int rez=0;

// Если съеден объект

if (mas[gY][gX]==-1) rez=1;

// Если перемещение в пустую ячейку

else if (mas[gY][gX]==0) rez=2;

// Если попадание в туловище змейки

else if (mas[gY][gX]>0) rez=3;

// В место перемещения головы

//установим значение - минус два

mas[gY][gX] = -2;

//Возвращаем результат

return rez;

}

Нам осталось добавить последний метод, который будет перемещать тело змейки. Метод будет называться perem(), именно этот метод вызывается таймером изменения логики игры.

Рассмотрим подробно и пошагово механизм перемещения змейки.Именно этот механизм будет выполняться при помощи метода perem(). Для этого нам придется проделать один шаг движения змейки. Пусть змейка перемещается влево:

Шаг 1.Исходное положение змейки (см. рис. 13)

Шаг 2.В новое положение головы помещается значение минус два: -2 (см. рис. 14)

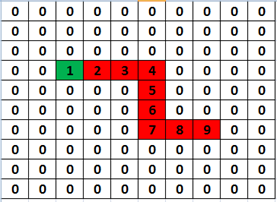


Рис. 13

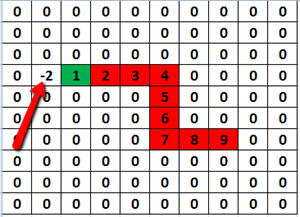


Рис. 14

Шаг 3.Пробегаем весь массив и все значения,которые больше нуля увеличиваем на единицу(см. рис. 15)

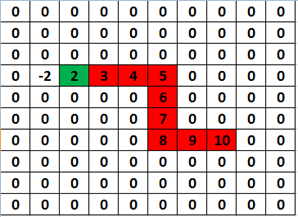


Рис. 15

Шаг 4.Вместо значения минус два: -2, помещаем единицу(см. рис. 16)

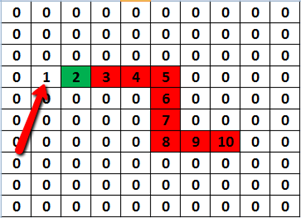


Рис. 16

Шаг 5

· Если при перемещении змейка съела объект, то больше ничего не делаем. Длина змейки уже увеличилась на единицу (см. рис. 16)

· Если при перемещении объекта не было, то число в массиве, которое больше длины змейки на единицу –заменяем нулевым(см. рис. 17)

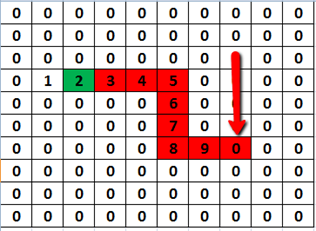


Рис. 17

Если этот алгоритм описать более кратко, то можно сказать, что голову змейки мы перемещаем по направлению движения, а хвост удаляем. В случае поедания объекта –хвост оставляем, и длина змейки увеличивается. При таком подходе змейка будет двигаться по нужному пути автоматически!

В класс game добавим метод, который выполняет описанный выше механизм:

// Перемещение змейки

public void perem()

{

// Перемещаем голову змейки

// и получаем результат перемещения

int flag = peremGolova();

// Если змейка врезалась в себя

// то признак окончания игры устанавливаем в истину

if (flag==3) endg=true;

// Два вложенных цикла, которые перебирают

// все элементы двухмерного массива

for (int i = 0; i < 30; i++)

{

for (int j = 0; j < 30; j++)

{

// Если значение больше нуля, то увеличиваем на 1

if (mas[i][j] > 0) mas[i][j]++;

// Если значение минус два,то меняем его на 1

else if (mas[i][j] == -2) mas[i][j]=1;

// Если не было поедания объекта

if (flag!=1)

{

// Если это хвост, то записываем в него нулевое значение

if (mas[i][j] == (dlina+1)) mas[i][j]=0;

}

}

}

// Если съеден объект

if (flag==1)

{

// Увеличиваем длину на единицу

dlina++;

// Генерируем новый объект для поедания

make\_new();

// Увеличиваем количество очков на 10

kol += 10;

}

//Изменяем направление движения

povorot();

}

Теперь класс game полностью закончен!И закончена реализация третьего уровня сложности! Запустим и посмотрим игру в действии (см. рис. 18):

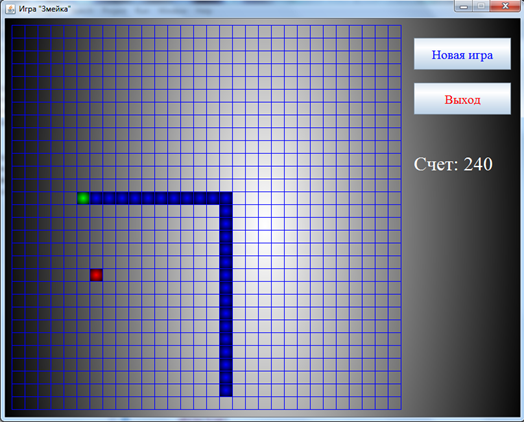


Рис. 18

Ниже представлен полный программный код файла game.java для третьего уровня сложности:

public class game

{

// Двухмерный массив для хранения игрового поля

public int[][] mas;

// Текущее направление движения

public int napr;

// Координаты головы змейки

private int gX, gY;

// Количество очков

public int kol;

// Новое направление движения

public int new\_napr;

// Длина змейки

private int dlina;

// Признак окончания игры

public boolean endg;

// Изменение направления движения змейки (поворот)

private void povorot()

{

if (Math.abs((new\_napr-napr))!=2)

{

napr=new\_napr;

}

}

// Конструктор класса

public game()

{

//Создаем новый массив 30x30

mas = new int[30][30];

}

// Генерация нового объекта в случайном месте

private void make\_new()

{

//Глухой (бесконечный цикл)

while(true)

{

// Получаем случайные значения x,y от 0 до 29

int x = (int)(Math.random()\*30);

int y = (int)(Math.random()\*30);

// Если в этом месте массива нулевое значение

// то помещаем туда объект для поедания змейкой

// и прерываем цикл

if (mas[y][x]==0)

{

mas[y][x] = -1;

break;

}

}

}

// Запуск игры (Старт)

public void start()

{

//Заполняем весь массив нулями

for (int i = 0; i < 30; i++) {

for (int j = 0; j < 30; j++) {

mas[i][j] = 0;

}

}

//Начальное направление - влево

napr = 0;

//Количество очков

kol = 0;

//Создаем начальную змейку с длиной три ячейки

mas[14][14] = 1;

mas[14][15] = 2;

mas[14][16] = 3;

dlina = 3;

//Записываем в переменные координаты головы змейки

gX = gY = 14;

//Признак окончания игры

endg=false;

//Формируем первый объект для поедания змейкой

make\_new();

}

// Перемещение головы змейки

public int peremGolova()

{

// Если текущее направление влево

if (napr == 0)

{

if ((gX - 1) >= 0)

gX--;

else

gX = 29;

}

// Если текущее направление вверх

else if (napr == 1)

{

if ((gY - 1) >= 0)

gY--;

else

gY = 29;

}

// Если текущее направление вправо

else if (napr == 2)

{

if ((gX + 1) <= 29)

gX++;

else

gX = 0;

}

// Если текущее направление вниз

else if (napr == 3)

{

if ((gY + 1) <= 29)

gY++;

else

gY = 0;

}

//Результат

int rez=0;

// Если съеден объект

if (mas[gY][gX]==-1) rez=1;

// Если перемещение в пустую ячейку

else if (mas[gY][gX]==0) rez=2;

// Если попадание в туловище змейки

else if (mas[gY][gX]>0) rez=3;

// В место перемещения головы

//установим значение - минус два

mas[gY][gX] = -2;

//Возвращаем результат

return rez;

}

// Перемещение змейки

public void perem()

{

// Перемещаем голову змейки

// и получаем результат перемещения

int flag = peremGolova();

// Если змейка врезалась в себя

// то признак окончания игры устанавливаем вистину

if (flag==3) endg=true;

// Два вложенных цикла, которые перебирают

// все элементы двухмерного массива

for (int i = 0; i < 30; i++)

{

for (int j = 0; j < 30; j++)

{

// Если значение больше нуля, то увеличиваем на 1

if (mas[i][j] > 0) mas[i][j]++;

// Если значение минус два,то меняем его на 1

else if (mas[i][j] == -2) mas[i][j]=1;

// Если не было поедания объекта

if (flag!=1)

{

// Если это хвост, то записываем в него нулевое значение

if (mas[i][j] == (dlina+1)) mas[i][j]=0;

}

}

}

// Если съеден объект

if (flag==1)

{

// Увеличиваем длину на единицу

dlina++;

// Генерируем новый объект для поедания

make\_new();

// Увеличиваем количество очков на 10

kol += 10;

}

//Изменяем направление движения

povorot();

}

}

Окончательный вариант игры осталось преобразовать в JAR-архив.

В классе myPanel при загрузке изображений необходимо изменить путь к изображениям. После создания JAR-архива изображения будут расположены в той же папке, что и JAR-архив. Поэтому необходимо прописать путь к текущей папке. Получится следующее:

// Попытка загрузки всех изображений для игры

try

{

fon = ImageIO.read(new File("./fon.png"));

telo = ImageIO.read(new File("./telo.png"));

golova = ImageIO.read(new File("./golova.png"));

ob = ImageIO.read(new File("./ob.png"));

endg = ImageIO.read(new File("./endg.png"));

}

catch (Exception ex) {}

"./" точка и наклонная черта означают путь к текущей папке.

После этого необходимо сохранить проект и обязательно запустить игру!

В процессе запуска вы увидите, что изображения не отобразились–это нормально, ведь в текущей папке изображений нет.

Нажимаем на заголовок проекта в общем списке проектов и нажимаем правую клавишу мыши. В открывшемся окне выбираем Export(см. рис. 19)

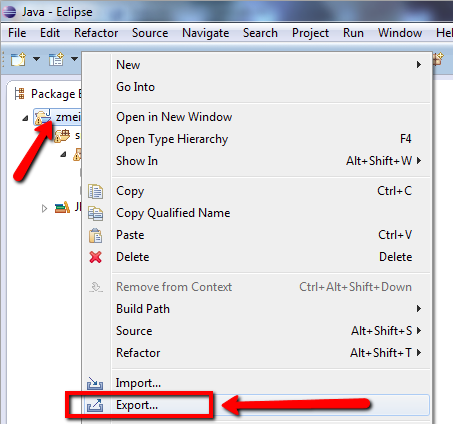


Рис. 19

В следующем окне Java, Runnable JAR file, Next (см. рис. 20):

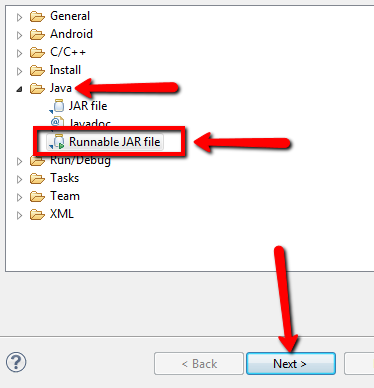


Рис. 20

В последнем окне в самой верхней строке необходимо выбрать:zmeika-zmeika,строкой ниже путь к JAR-архиву и имя: zmeika.jar. Переключатель устанавливаем в среднее положение и нажимаем Finish (см. рис.21).

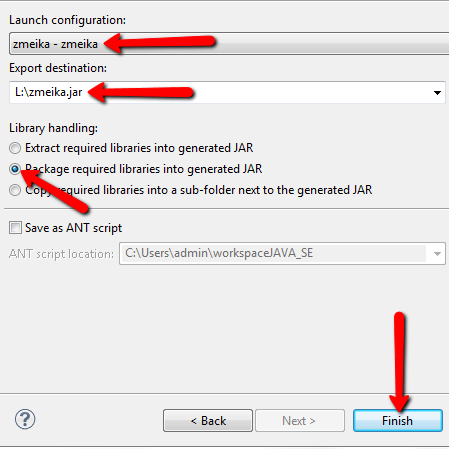


Рис. 21

Последним шагом создаем папку с именем zmeikaи помещаем в нее все пять изображений и сам JAR-архив (см. рис.22).

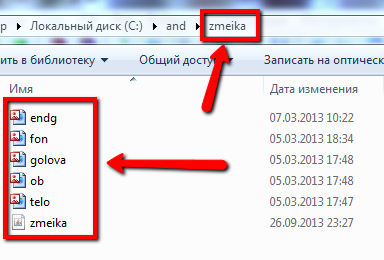


Рис. 22

Папку zmeika c пятью изображениями иJAR-архивом можно считать конечным продуктом для передачи пользователю.