**Практическая работа 4.3. Реализация первого уровня создания змейки.**

Сначала определим схему классов, которые потребуются при создании игры. Всего для создания игры нам потребуется четыре класса .

После создания проекта мы добавим к нему два файла: zmeika.java и game.java. Классы zmeika, myFrame , myPanel будут находиться в файле zmeika.java; класс game будет находиться в файле game.java.Другими словами в первом файле будут находиться сразу три класса, которые отвечают за запуск игры и ее графическое представление. Класс, отвечающий за логику игры, будет находиться в отдельном файле.

Рассмотрим задачи,выполняемые каждым классом:

1. Класс zmeika, который является главным классом приложения, будет выполнять запуск игры, в нем будет создаваться объект окна для игры и в нем, конечно же, будет метод main().

2. Класс myFrame –это класс окна,который наследуется от класса JFrame. В нем будет подключаться панель окна, на которой расположено игровое поле. Также в этом классе будут указаны все параметры окна:его размеры, заголовок и т.д.

3. Класс myPanel –это класс панели окна, который наследуется от класса JPanel.Этот класс представляет собой игровое поле. Именно этот класс будет подключаться к окну в классе myFrame. Вся графика (отрисовка игрового поля), настройка таймеров, обработка клавиатуры будет происходить именно в этом классе. Но все данные для отрисовки игрового поля класс myPanelбудет получать из класса game, в котором находиться двумерный массив.

4. Класс game– это класс для реализации логики игры. В этом классе будет находиться двумерный массив, и все операции с двумерным массивом будут производиться именно в этом классе. КлассmyPanel будет постоянно обращаться к классу game, чтобы выполнить отрисовку игрового поля по данным двухмерного массива. Данные массива будут зеркально отображаться на игровом поле в классе myPanel, но сами изменения данных массива будут происходить в классе game.

Такая схема классов,конечно же, не является единственной для создания игры Змейка. Программист сам определяет для себя схему классов и их название.

Алгоритм игры,использованный при реализации игры Змейка, также не является единственным. Программист может использовать алгоритм по своему усмотрению. На начальном этапе программирования не стоит гнаться за малым объемом программного кода. Главное больше практики! Полный вариант игры Змейка,представленный в данном методическом пособии (если убрать из него пустые строки и комментарии) в сумме будет составлять250 строк программного кода.С вашим опытом объем программного кода будет уменьшаться!

Прежде всего, подготовим необходимые изображения, которые понадобятся при создании игры. На этапе разработки можно взять простые изображения, сделанные в любом графическом редакторе и писать программный код с их использованием. Вполне подойдут картинки,найденные в интернете или сделанные самостоятельно. Когда игра будет закончена, можно сделать более внушительный дизайн, заменив исходные варианты картинками такого же размера, но с более привлекательным дизайном. Вспомним,что основной формат при разработке наJava –это формат PNG. Именно в этом формате необходимо подготовить изображения.

Выполнить конвертацию в формат PNG можно во многих графических редакторах.

Для игры Змейка понадобятся пять изображений:

1) Основной фон

2) Объект для поедания Змейкой

3) Голова Змейки

4) Элемент тела Змейки

5) Изображение Конец Игры

Подготовим следующие изображения:

· fon.png (см. рис. 2) –это фоновое изображение, размер 800x650 –размеры окна по условию задания.

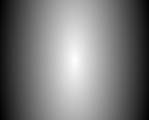


Рис. 2

· telo.png (см. рис. 3) –элемент тела Змейки, размер 20x20– размер одной клетки игрового поля по условию задания.

http://ok-t.ru/life-prog/baza2/334145791143.files/image102.png

Рис. 3

· golova.png (см. рис. 4) –голова Змейки, размер 20x20 –размер одной клетки игрового поля по условию задания.

http://ok-t.ru/life-prog/baza2/334145791143.files/image103.png

Рис. 4

· ob.png (см. рис. 5) –объект для поедания Змейкой, размер 20x20 –размер одной клетки игрового поля по условию задания.

http://ok-t.ru/life-prog/baza2/334145791143.files/image104.png

Рис. 5

· endg.png (см. рис. 6) –изображение "Конец Игры", размер 200x100–размер взят произвольно, здесь главное, чтобы рисунок помещался на игровом поле.

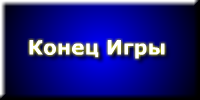


Рис. 6

Все пять изображений поместим, например, в корень диска C:\. В процессе разработки игры изображения будут загружаться из корня диска C:\. После окончания создания игры мы поместим их внутрь JAR-архива.

Откроем среду разработки Eclipse для Java и создадим новый проект Java. Для этого в меню выберем: File-New-Java Project (см. рис. 7)

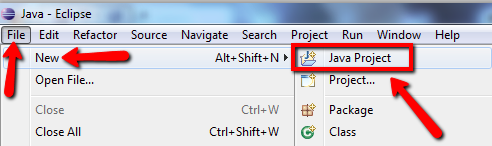


Рис. 7

Укажем название проекта: zmeika (см. рис. 8) и нажмем кнопку Finish(Готово).

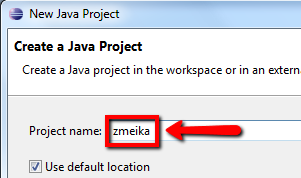


Рис. 8

В списке проектов добавится новый проект zmeika. Добавим к нему новый класс.Для этого нажмем на имя проекта левой клавишей мыши (выделим). Далее нажмем правую клавишу мыши, откроется дополнительное окно. Выберем New-Class(см. рис. 9):

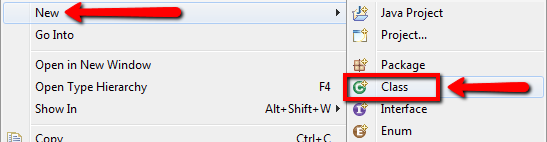


Рис. 9

Это будет главный класс нашего проекта,в нем будет находиться метод main(). Укажем название класса: zmeika и поставим галочку public static void main(см.рис. 10):

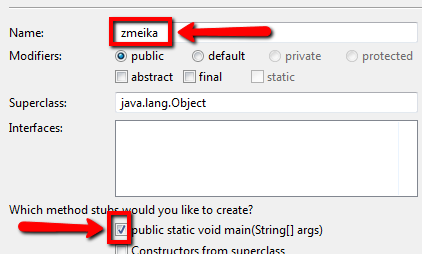


Рис. 10

Далее нажимаем кнопку Finish (Готово). В результате появится программный код нового класса, содержащий метод main() (см. рис. 11):

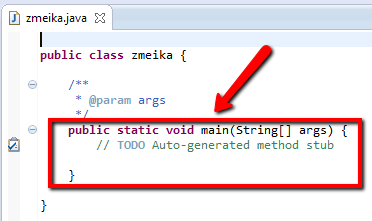


Рис. 11

При разработке игры Змейка мы не будем использовать визуальный конструктор, потому что интерфейс довольно простой. Элементы интерфейса - кнопки и надпись мы создадим программно. На первом занятии мы выяснили, что программирование игры Змейка можно свести к программированию двумерного массива. Таймер отрисовки будет отображать двумерный массив, перенося его значения на игровое поле. Все объекты игрового поля будут представлены в массиве в виде чисел. Создание игры можно разбить на две основные части –это различные действия с массивом для перемещения змейки и графическое отображение игрового поля.

Для операций с массивом создадим еще один класс. Добавим к проекту еще один класс с названием: game (см.рис. 12), галочку public static void main() на этот раз ставить не нужно! (см. рис.12).

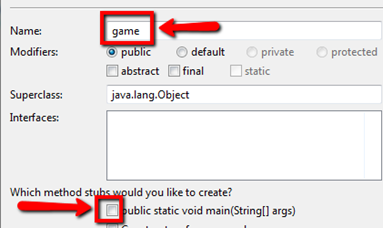


Рис. 12

В результате в нашем проекте будет два файла (см.рис. 13, 14):

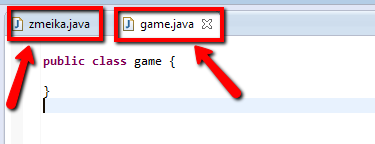


Рис. 13

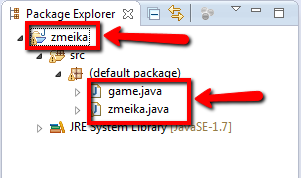


Рис. 14

В файле game.java будет реализована вся логика игры –это операции с двумерным массивом. будет реализована вся графика, построение интерфейса, обработка событий, отрисовка и настройка таймеров.

Начнем с класса game. Для первого уровня сложности достаточно выводить по центру только голову змейки, которая не перемещается и объект для поедания,появляющийся в случайном месте.

Сначала добавим свойства класса, которые нам понадобятся при работе с массивом. Нам будет достаточно только двумерного массива. Итак, начнем писать программный код в файле game.java:

{public class game

// Двумерный массив для хранения игрового поля

public int[][] mas;

}

Объявление int[][] mas; означает двумерный массив целых чисел. Свойство класса –это переменная класса. Данную переменную класса мы сделали public –открытой. Это необходимо для доступности массива из другого класса. К данным массива мы будем обращаться для отрисовки данных игрового поля. Добавим конструктор класса:

public class game

{

// Двухмерный массив для хранения игрового поля

public int[][] mas;

// Конструктор класса

public game()

{

//Создаем новый массив 30x30

mas = new int[30][30];

}

}

Конструктор класса автоматически вызывается при создании объекта на основании класса. Имя конструктора должно совпадать с именем класса.

В конструкторе мы создаем двумерный массив размером 30x30,тридцать строк и тридцать элементов в каждой строке. Именно такие размеры имеет игровое поле для змейки. Когда происходит объявление массива, массива еще нет. Он создается при помощи команды new. При объявлении массива мы лишь сообщаем, что тип данных–двумерный массив.

Далее нам понадобится метод для генерации положения объекта для поедания змейкой. Внутри (под конструктором класса) класса, добавим метод make\_new():

// Генерация нового объекта в случайном месте

private void make\_new()

//Глухой (бесконечный цикл)

c

//Глухой (бесконечный цикл)

while(true)

{

}

Цикл будет прерван в тот момент, когда будет найдена ячейка с нулевым значением:

// 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;

// Прерываем цикл

break;

}

Перед выходом из цикла записываем в эту ячейку минус единицу -"объект для поедания". Для поиска случайной ячейки на игровом поле используем следующую конструкцию:

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

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

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

Для получения искомой ячейки нужно выбрать случайную строку и случайный номер элемента в строке. И эти значения (номер строки и номер элемента в строке) должны лежать от 0 до 29 включительно. Нумерация в массиве начинается с нуля. Последний метод, который нам понадобится в этом классе – это метод для старта(запуска) игры. Под методом make\_new()добавим метод start():

// 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;

}

}

//Размещаем голову змейки по центру поля

mas[15][15] = 1;

//Формируем первый объект для поедания змейкой

make\_new();

}

Этот метод содержит действия при начале игры. Сначала обнуляем весь массив при помощи двух вложенных циклов –это равноценно полному очищению игрового поля.Далее по центру игрового поля размещаем голову змейки (помещаем единицу):

//Размещаем голову змейки по центру поля

mas[15][15] = 1;

Теперь генерируем случайным образом "объект для поедания",вызывая метод make\_new(), созданный на предыдущем шаге.Полностью программный код класса game public class game

{

для первого уровня сложности выглядит так:

// Двухмерный массив для хранения игрового поля

public int[][] mas;

// 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;

}

}

//Размещаем голову змейки по центру поля

mas[15][15] = 1;

//Формируем первый объект для поедания змейкой

make\_new();

}

}

Перейдем в другой файл нашего проекта: zmeika.java.В этом файле будет реализована остальная часть игры. В самом верху добавим необходимые библиотеки:

// 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.\*;

Все эти библиотеки нам уже знакомы по игре "Новогодний дождь".Внутри метода main(), который является точкой входа в программу, добавим создание объекта "окно"с игровым полем:

// Главный класс программы

public class zmeika

{

// Метод запуска приложения

public static void main(String[] args)

{

// Создание объекта окна игрового поля

myFrame okno = new myFrame();

}

}

Сам класс myFrame будет создан на следующем шаге и будет располагаться под классом zmeika:

// 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 myFrame наследуется от класса JFrame –класса окон. Внутри класса myFrame находится конструктор класса, который задаёт все характеристики окна. Все эти действия мы выполняли при создании игры "Новогодний дождь". Отдельного внимания заслуживает строка:

// Запрет изменения размеров окна

setResizable(false);

Она запрещает пользователю изменять размеры окна по ширине и высоте.

При задании заголовка окна слово "Змейка" находится внутри двойных кавычек, поэтому перед кавычками стоит наклонная черта \.:

//Заголовок окна

setTitle("Игра \"Змейка\"");

Следующие строки подключают панель к окну:

// Создание объекта панели и подключения ее к окну

myPanel pan = new myPanel();

Container cont = getContentPane();

cont.add(pan);

Класс myPanel будет располагаться под классом myFrame. Этот класс реализует игровое поле и будет самым большим по объему программного кода.Создадим класс myPanel и добавим свойства (переменные)класса:

// Класс панели игрового поля

class myPanel extends JPanel

{

// Переменная для реализации логики игры

private game myGame;

// Таймер отрисовки

private Timer tmDraw;

// Изображения, используемые в игре

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

// Надпись для количества очков

private JLabel lb;

// Две кнопки

private JButton btn1,btn2;

}

Среди свойств представлено все, что понадобится нам на игровом поле:

· переменная типа game(класс, созданный в самом начале для реализации двумерного массива),

· таймер для отрисовки,

· пять изображений для игры,

· текстовая надпись,

· две кнопки.

Добавим конструктор класса myPanel:

// Конструктор класса

public myPanel()

{

//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();

//Запускаем игру

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();

setLayout(null);

lb = new JLabel("Счет: 0");

lb.setForeground(Color.WHITE);

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

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

add(lb);

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() {

public void actionPerformed(ActionEvent arg0) {

myGame.start();

}

});

add(btn1);

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() {

public void actionPerformed(ActionEvent arg0) {

System.exit(0);

}

});

add(btn2);

}

Рассмотрим представленный выше программный код подробно.

Вначале производим загрузку всех пяти изображений:

//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) {}

Затем создаем объект на основании класса game и производим запуск игры:

//Создаем объект новой игры

myGame = new game();

myGame.start();

В этот момент создается двумерный массив 30x30и в него вписываются значения при запуске игры. После этого настраиваем таймер для отрисовки:

// 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();

Таймер вызывает метод repaint() и перерисовывает игровое поле. Скорость срабатывания таймера –20 миллисекунд (50 раз в секунду). Теперь приступаем к добавлению элементов интерфейса и включаем возможность произвольного размещения:

setLayout(null);

Добавляем текстовую надпись, которая показывает счет игры:

lb = new JLabel("Счет: 0");

lb.setForeground(Color.WHITE);

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

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

add(lb);

Указываем цвет, шрифт,расположение и в конце добавляем ее к панели!

Затем добавляем кнопкуНовая 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() {

public void actionPerformed(ActionEvent arg0) {

myGame.start();

}

});

add(btn1);

К кнопке подключается обработчик события при нажатии на кнопку.При нажатии вызывается метод: myGame.start(), который сбрасывает значения в массиве на начальные, и игра перезапускается. На последнем шаге в конструкторе добавляется вторая кнопка - 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() {

public void actionPerformed(ActionEvent arg0) {

System.exit(0);

}

});

add(btn2);

В обработчике события при нажатии на эту кнопку вызывается System.exit(0), что приводит к завершению работы приложения.Осталось добавить последний метод к классу myPanel. Для этого ниже конструктора класса записываем следующий код:

// Метод отрисовки

public void paintComponent(Graphics gr)

{

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)

{

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

}

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

{

gr.drawImage(ob,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++)

{

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

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

}

}

Метод paintComponent()отвечает за рисование всей графики. Этот метод мы многократно использовали ранее. В первую очередь рисуется фон игрового поля. Далее идут два вложенных цикла, которые перебирают все элементы двухмерного массива. Если значение нулевое –ячейка остается пустой, если значение равно-1, то выводится"объект для поедания", если значение равно 1,то выводится голова змейки.

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

Метод drawImage() выводит изображение. Первый параметр –это переменная, в которую загружена картинка, второй и третий параметр –это координаты верхней левой угловой точки картинки, четвертый и пятый параметр – это ширина и высота картинки, последний параметр –пустое значение. Обратите внимание, что второй и третий параметр представляют из собой формулу. В эту формулу подставляются значения переменных i, j.

Чтобы понять, как работает подобная формула –можно взять лист бумаги и самому выполнить алгоритм пошагово, подставляя значения переменных i,j. Тогда можно увидеть значения, которые увеличиваются с шагом в 20 пикселей –это размер одной клетки. На последнем шаге в методе рисуется сетка из линий:

s

Цикл рисует вертикальные и горизонтальные линии. В методе drawLine()четыре параметра –первые два –это x,y начальной точки линии, вторые два–это x,y конечной точки линии. Цикл можно пройти пошагово и подставить значения переменной i:

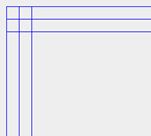


Рис. 15

Первый шаг цикла, i=0:

gr.drawLine(10, 10, 10, 610);

gr.drawLine(10, 10, 610, 10);

Второй шаг цикла, i=1:

gr.drawLine(30, 10, 30, 610);

gr.drawLine(10, 30, 610, 30);

Третий шаг цикла, i=2:

gr.drawLine(50, 10, 50, 610);

gr.drawLine(10, 50, 610, 50);

Можно увидеть закономерность –первая линия сдвигается слева направо, а вторая линия сдвигается сверху вниз. Сдвиг происходит с шагом 20 пикселей.Рассмотренные три шага цикла нарисуют линии (см. рис. 15). На этом написание программного кода в файле zmeika.java закончено. Полностью программный код этого файла представлен ниже:

// 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;

// Изображения, используемые в игре

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

// Надпись для количества очков

private JLabel lb;

// Две кнопки

private JButton btn1,btn2;

// Конструктор класса

public myPanel()

{

//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();

setLayout(null);

lb = new JLabel("Счет: 0");

lb.setForeground(Color.WHITE);

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

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

add(lb);

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() {

public void actionPerformed(ActionEvent arg0) {

myGame.start();

}

});

add(btn1);

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() {

public void actionPerformed(ActionEvent arg0) {

System.exit(0);

}

});

add(btn2);

}

// Метод отрисовки

public void paintComponent(Graphics gr)

{

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)

{

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

}

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

{

gr.drawImage(ob,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++)

{

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

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

}

}

}

Создание игры Змейка для первого уровня сложности на этом закончено. Запустим наш проект, в результате должно получиться следующее(см. рис. 16):

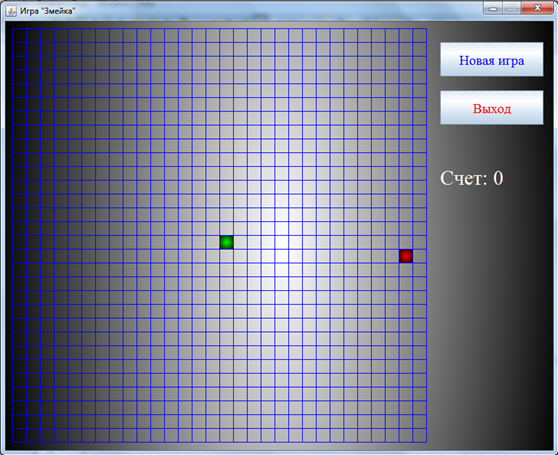


Рис. 16