Санкт-Петербургский государственный университет информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

М.С. Богданов, А.А. Шалыто

Компьютерная игра

Lode Runner

Проектная документация

Проект создан в рамках
«Движения за открытую проектную документацию»
http://is.ifmo.ru

Санкт-Петербург
2005

Содержание

Введение

Известна игра *Lode Runner* для компьютеров *Sinclair* [1]. Игра состоит в том, что игрок, управляя героем, собирает в некотором мире золото, а ему противодействуют враги. Правила этой игры приведены в следующем разделе.

В Интернете можно найти также большое число её клонов [2, 3]. Однако указанные разновидности игры не содержат проектной документации и исходных кодов.

Цель настоящей работы – разработка проекта игры и его реализация. Для описания поведения героя используются конечные автоматы, позволяющие упростить написание логики программы.

При построении логики программы используется подход, предложенный в работе [4]: состояния программы разделяются, как и в машине Тьюринга, на управляющие и вычислительные. При этом небольшое число управляющих состояний может управлять сколь угодно большим числом вычислительных состояний.

В этом случае граф переходов строится для управляющих состояний *персонажа*, которые задают логику игры. В управляющем состоянии вычисляется та или иная функция, обеспечивающая его движение. Переменные, используемые в этой функции, естественно могут находиться в сколь угодно большом числе состояний. В каждом управляющем состоянии *персонаж* может находиться в огромном числе вычислительных состояний.

Отметим, что в игре используются *персонажи* двух типов: *герой* и *враг*, поведение которых описывается аналогичными графами переходов, отличающимися источниками входных воздействий. При этом *герой*, естественно, один, а *врагов* – много.

Программа написана в виде аплета на языке *Java* в среде *NetBeans 3.5.1*. Аплет и исходные коды программы приведены на сайте http://is.ifmo.ru в разделе “Проекты”.

1. Правила игры

- Игрок, управляя *героем*, должен собрать все золото на рассматриваемом уровне. После этого должна появиться лестница на выход. При достижении верхнего элемента лестницы *герой* переходит на следующий уровень, если он существует. В противном случае выдаются результаты игры, либо информации о том, что игра закончена.

- Игровое поле представляет собой решетку, состоящую из ячеек. Размер каждой ячейки 16 x 16 пикселей.

- *Герой* и *враги* могут перемещаться по лестнице, веревке, кирпичам и бетону:
  - при движении по лестнице возможны два режима:
    - влево/вправо, если *герой* находится в центре ячейки;
    - вверх/вниз;

- кирпичи могут быть выбиты *героем*, образуя яму, которая через некоторое время может «зарастать», превращаясь снова в кирпич;
- если *герой* или *враг* попадает в зарастающую яму, то он погибает;
- *враги* через некоторое время после гибели снова появляются на рассматриваемом уровне;
- *герой* может свободно перемещаться внутри нескольких расположенных рядом ям;
- *враг* не может делать ямы и, попадая туда, он проводит в ней некоторое время, после чего выбирается из нее, если, конечно, яма не начала зарастать.

Приведем изображения основных элементов игры:

-  – кирпич;
-  – бетон;
-  – лестница;
-  – веревка;
-  – золото.

Отметим, что черный цвет в этих элементах – фон.

И типичный вид уровня, составленный из использованных элементов – рис. 1, 2:

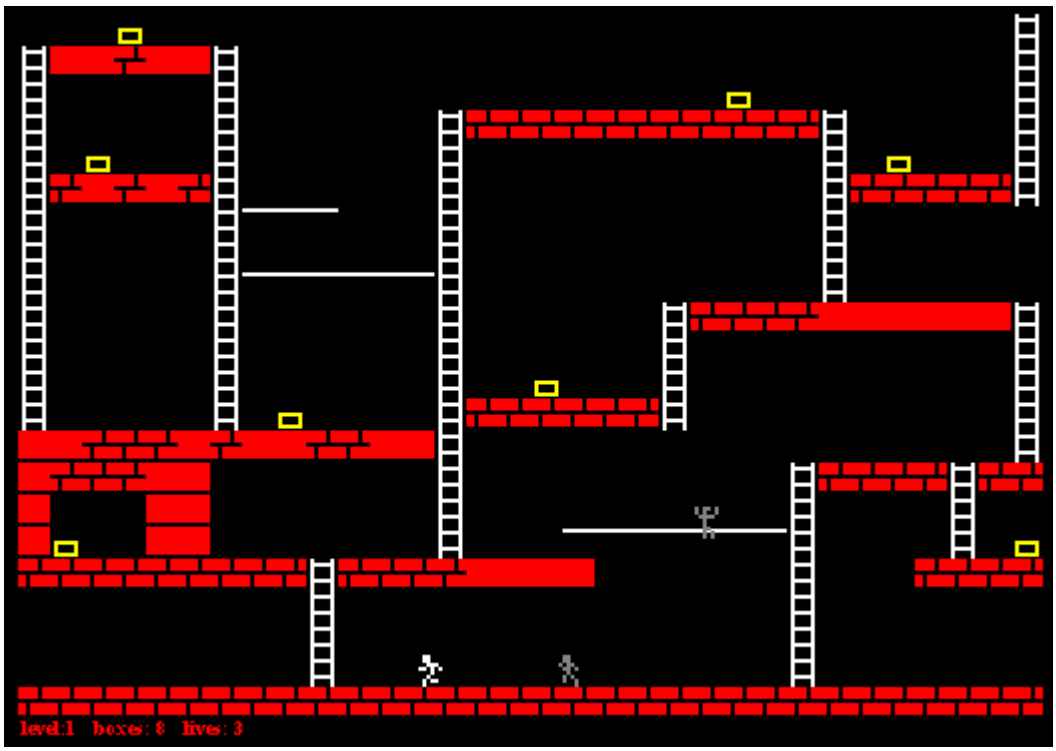

Рис.1. Второй уровень

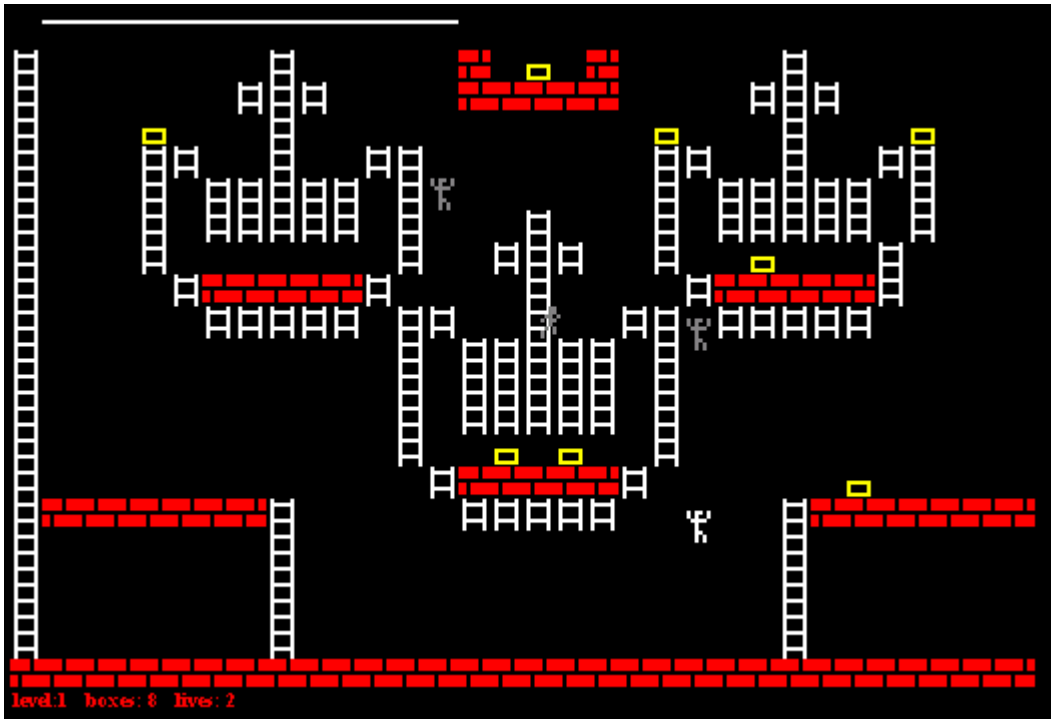

Рис. 2. Четвертый уровень

На этих рисунках герой – белый, а враги – темные.

2. Диаграмма классов

На рис. 3 приведена диаграмма классов, построенная для рассматриваемой игры.

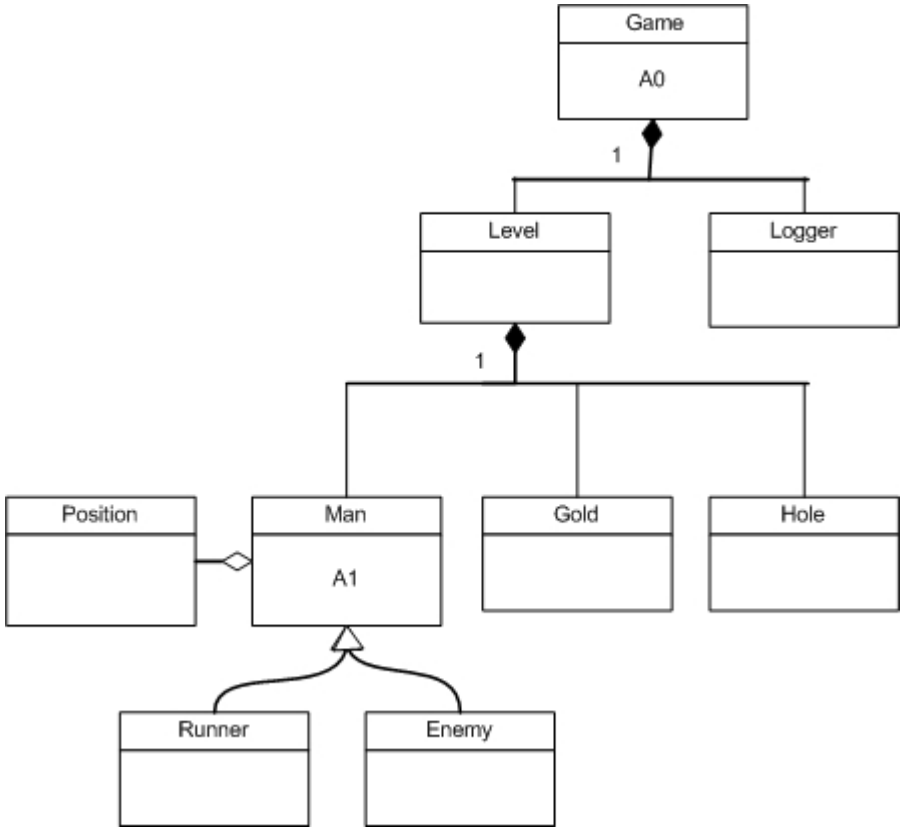

Рис. 3. Диаграмма классов

Приведем краткую характеристику используемых классов:

- класс *Game* обеспечивает взаимодействие с игроком (автомат *A0*);
- класс *Level* обеспечивает загрузку уровней, а также содержит все данные, характеризующие уровень;
- класс *Man* содержит общие методы классов *Runner* и *Enemy*, визуализирует их, содержит автомат *A1*, обеспечивающий изменения управляющих состояний как *героя*, так и *врагов*;
- класс *Position* является полем класса *Man* и хранит координаты *персонажа*;
- класс *Enemy* определяет путь до *героя* и вызывает автомат *A1* для передвижения по этому пути;
- класс *Runner* является наследником класса *Man* и содержит информацию о собранном золоте, игровых очках и оставшихся жизнях *героя*;
- классы *Lode* и *Hole* содержат данные о структурах таких элементов игры как золото и яма, а также визуализируют эти элементы;
- класс *Logger* обеспечивает протоколирование.

3. Класс *Game*

3.1. Словесное описание

Класс *Game* является наследником стандартного *Java*-класса *Applet* и определяет область визуализации внутри браузера. Обрабатывает сообщения клавиатуры и при необходимости передает их автомату *A1* класса *Man*, вызывая его. Этот класс обеспечивает взаимодействие с игроком. Визуализацию обеспечивает автомат *A0*, являющийся одним из методов рассматриваемого класса.

3.2. Автомат *A0*

На рис.4 приведена схема связей автомата *A0*.

Рис. 4. Схема связей автомата *A0*

На рис. 5 приведен граф переходов автомата *A0*.

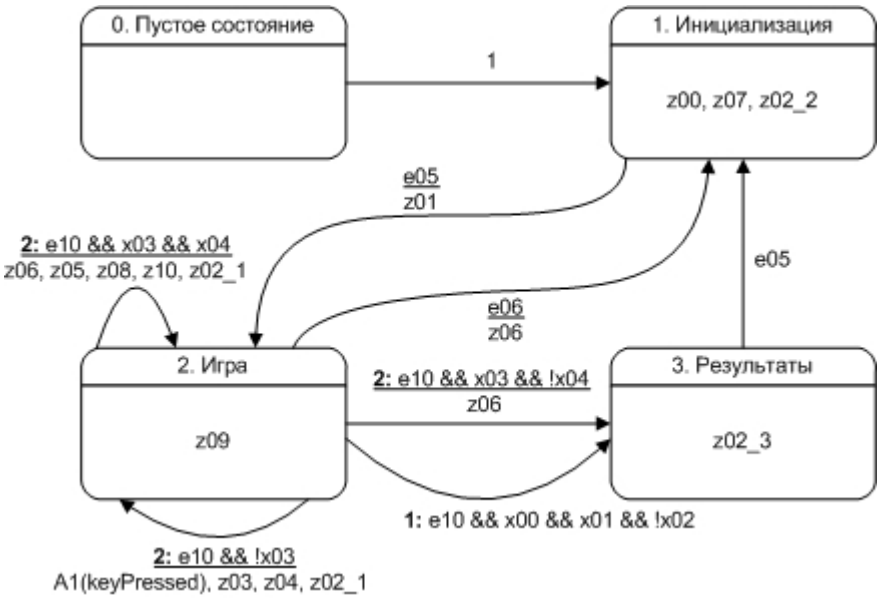

Рис. 5. Граф переходов автомата *A0*

4. Класс *Man*

4.1. Словесное описание

Класс *Man* является базовым для классов *Runner* и *Enemy*.
В зависимости от поступающих сигналов управления соответствующим образом меняет состояние героя/врага, а также при необходимости изменяет карту уровня. В каждый момент времени герой/враг находится в какой-либо ячейке, причем в каждой ячейке он может находиться в в трех различных положениях как по горизонтали , так и по вертикали: Z – в центре ячеки по x/y, W – слева от центра, E - справа от центра, S – снизу от центра, N – сверху от центра (соответственно SE – снизу справа). При передачи параметров процедурам движения используются те же обозначения S – вверх, N – вниз, W – влево, E – вправо. Sn означает S, SW или SE в зависимости от того, где находится персонаж: в центре ячейки, слева или справа от нее, соответственно.

4.2. Автомат *A1*

На рис. 6 приведена схема связей автомата *A1*.

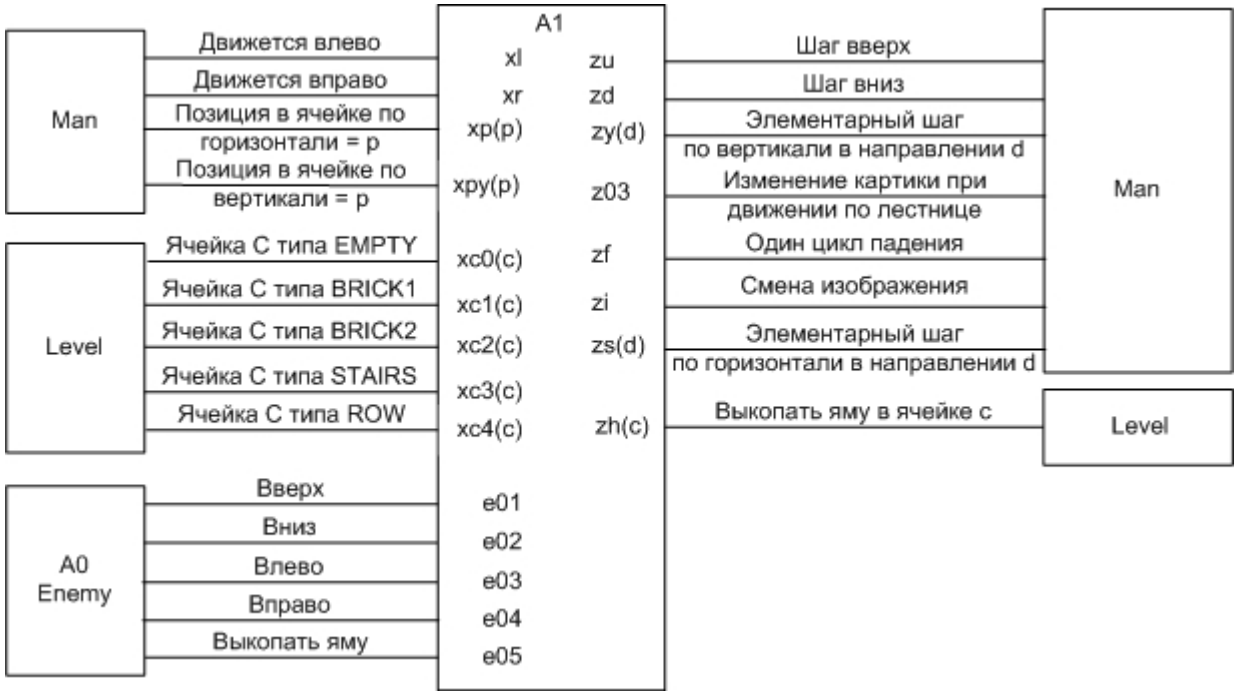

Рис. 6. Схема связей автомата *A1*

На рис.7 приведен граф переходов автомата *A1*.

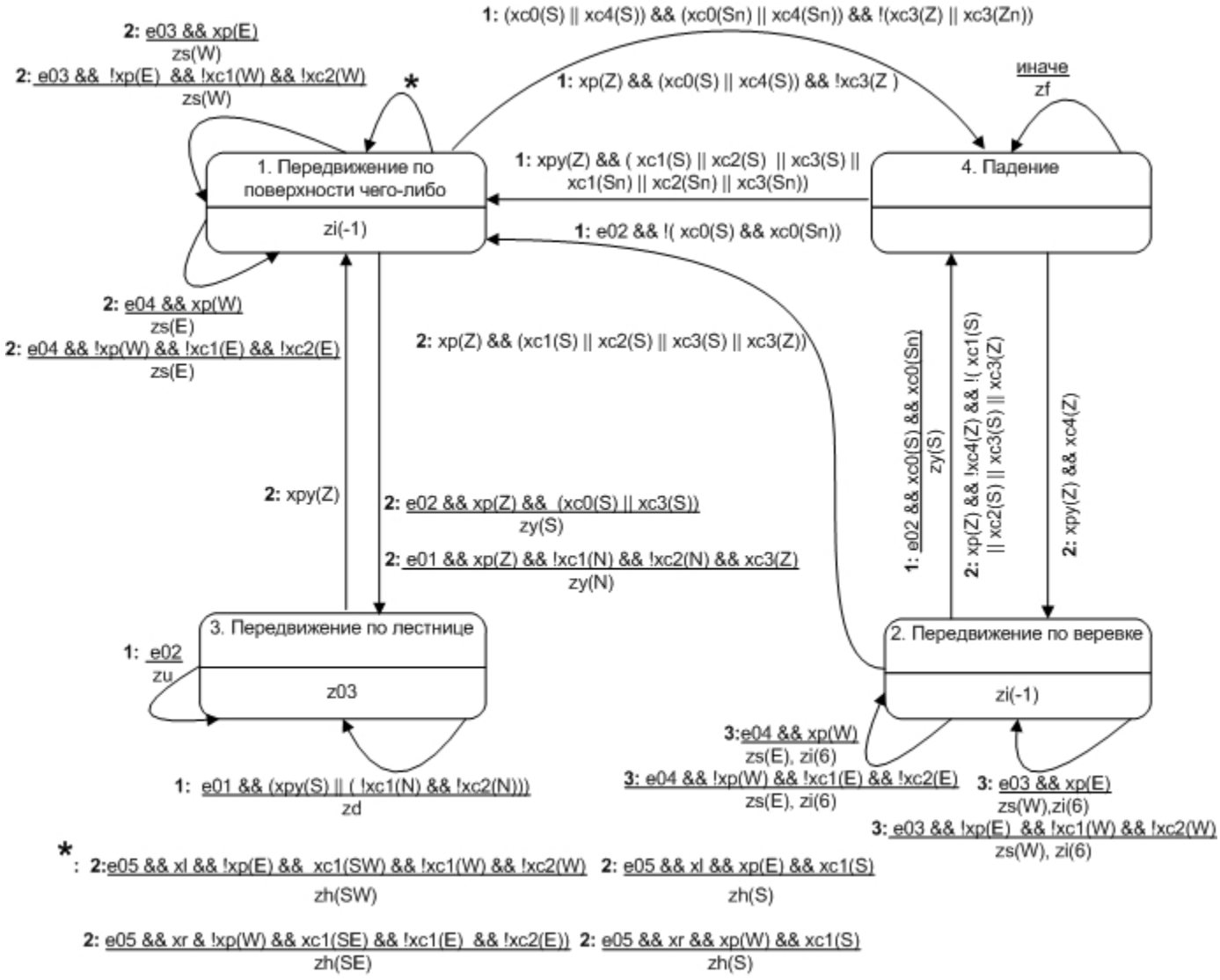

Рис. 7. Граф переходов автомата *A1*

5. Классы Enemy и Level

Класс *Enemy* обеспечивает автоматическое управление *врагом*. Это делается на основе матрицы минимальных длин путей из каждой ячейки в ячейку, в которой в текущий момент находится *герой*: в каждой ячейке хранится минимальное количество шагов, которые необходимо сделать, чтобы добраться до *героя*. На каждом шаге *враг* переходит из текущей ячейки в соседнюю с меньшим значением в матрице минимальных длин путей. Для этого осуществляется обход всей карты процедурой, приведенной ниже, чем-то напоминающий процедуру обхода графа в ширину, из ячейки, соответствующей текущей позиции *героя*. Процедура извлекает из списка *gray* первый элемент, в котором хранятся координаты ячейки, которую в дальнейшем будем называть ячейкой **a,** и смотрит можно ли в нее попасть из соседней ячейки, которую в дальнейшем будем называть ячейкой **b**. Если это возможно, то в

список добавляется элемент, содержащий координаты ячейки **b**, а в матрицу минимальных длин путей для ячейки **b** записывается значение из ячейки **a + 1**.

Описанный фрагмент процедуры реализуется следующим образом:

```
while (gray.size() > 0) {
   Object o = gray.remove(0);
   if (o != null) {
       y = ((Integer) o).intValue() / 32;
        x = ((Integer) o).intValue() % 32;

       if ((getSBoxAT(y - 1, x) != BRICK1) && (getSBoxAT(y - 1, x) != BRICK2) &&
   (grayMap[y - 1][x] == grayINF)) {

          gray.add(new Integer( (y - 1) * 32 + x));
          grayMap[y - 1][x] = (byte)(grayMap[y][x] + 1);
       }

       if  (((getSBoxAT(y, x + 1) == STAIRS) || (getSBoxAT(y, x + 1) == ROW) ||
   (((getSBoxAT(y + 1, x + 1) == BRICK1) || (getSBoxAT(y + 1, x + 1) ==
   BRICK2)||(getSBoxAT(y + 1, x + 1) == STAIRS)) && (getSBoxAT(y, x + 1) == EMPTY)) ) &&
   (grayMap[y][x + 1] == grayINF)) {

          gray.add(new Integer( y * 32 + x + 1));
          grayMap[y][x + 1] = (byte)(grayMap[y][x] + 1);
       }

       if ((getSBoxAT(y + 1, x) == STAIRS) && (grayMap[y + 1][x] == grayINF)) {

          gray.add(new Integer( (y + 1) * 32 + x));
          grayMap[y + 1][x] = (byte)(grayMap[y][x] + 1);
       }

       if  (((getSBoxAT(y, x - 1) == STAIRS) || (getSBoxAT(y, x - 1) == ROW) ||
   (((getSBoxAT(y + 1, x - 1) == BRICK1) || (getSBoxAT(y + 1, x - 1) == BRICK2) ||
   (getSBoxAT(y + 1, x - 1) == STAIRS)) && (getSBoxAT(y, x - 1) == EMPTY))) &&
   (grayMap[y][x - 1] == grayINF))  {

          gray.add(new Integer( y * 32 + x - 1));
          grayMap[y][x - 1] = (byte)(grayMap[y][x] + 1);
       }
   }
}
```

Значение в матрице минимальных длин путей пересчитываются не на каждом шаге, а с задержкой. Поэтому *враги* могут и не преследовать *героя*, а бежать в другую сторону.

Существуют и другие варианты реализации автоматического управления *врагом*: для каждого врага можно, например, осуществлять поиск в ширину из его текущей ячейки до обнаружения ячейки с координатами *героя* и запоминать путь, найденный при поиске, однако, предложенный вариант кажется эффективнее.

Процедура в целом реализуется в классе *Level.* Он также обеспечивает доступ к текущему состоянию карты уровня и производит его загрузку, инициализирует *врагов* и *героя*.

Программа в целом приведена в приложении 1, а в приложении 2 – протокол работы системы в начале игры.

Заключение

Как показывает опыт написания проекта разделение состояний на управляющие и вычислительные позволяет выделить “слабо” связанные куски кода и уже в них реализовывать специфическое поведение. При традиционном подходе это было бы затруднительно, и пришлось бы писать множество мало понятных вложенных операторов `if`.

Источники

1. *Эмулятор* ***ZX-Emul*** (http:// lion17home.narod.ru), код игры на http://www.worldofspectrum.org.
2. http://gamer.raduga.ru/loder/index.shtml
3. http://b-timka.narod.ru/loderunner/
4. *Шалыто А.А., Туккель Н.И.* От тьюрингова программирования к автоматному // Мир ПК. 2002. № 2, с.144 – 149 (http://is.ifmo.ru, раздел «Статьи»).

Приложение 1. Листинги программ

1. Класс *Game*

```
package loderunner;

import java.awt.*;
import java.applet.Applet;
import java.io.*;
import java.util.*;

import java.awt.event.*;


public class Game extends Applet implements KeyListener {

    final static int e01 = 01;      // нажата клавиша UP
    final static int e02 = 02;      // нажата клавиша DOWN
    final static int e03 = 03;      // нажата клавиша LEFT
    final static int e04 = 04;      // нажата клавиша RIGHT
    final static int e05 = 05;      // нажата клавиша SPACE/HOLE
    final static int e06 = 06;      // нажата клавиша ESC
    final static int e00 = 00;      // нажата другая клавиша
    final static int e10 = 10;      //событие от таймера

    // Состояния автомата A0
    private final int y00 = 0;
    private final int y01 = 1;
    private final int y02 = 2;
    private final int y03 = 3;
    private int state = y00; // начальное состояние

    private final int infoHeight = 10;
    private final Font font = new Font("Serif", Font.PLAIN|Font.ITALIC, 30);
    private Image buffer,sbuffer;
    private Level level;
    private Image picture;
    private Image rImages[];
    private Image eImages[];
    private Image coreImages[];
    public static Image holeImages[];
    public static Image lode;

    private Man [] men;
    private int levelIndex;
    private int keyPressed;
    private Timer timer;
    private boolean isWaiting  = false;

    public static Logger logger = new Logger();

    private void initBuffer(int w,int h){
        buffer = createImage (w, h);
    }

//  Загрузка изображений
    public void loadImages() {
        MediaTracker mt;
```

```
mt = new MediaTracker(this);

coreImages = new Image[4];
coreImages[0] = getImage(getDocumentBase(),"Images/brick1.gif");

coreImages[1] = getImage(getDocumentBase(),"Images/brick2.gif");

coreImages[2] = getImage(getDocumentBase(),"Images/stair.gif");
coreImages[3] = getImage(getDocumentBase(),"Images/clue.gif");

rImages = new Image[15];
rImages[0] =  getImage(getDocumentBase(),"Images/runner0.gif");

rImages[1] =  getImage(getDocumentBase(),"Images/runner-right1.gif");

rImages[2] =  getImage(getDocumentBase(),"Images/runner-right2.gif");

rImages[3] =  getImage(getDocumentBase(),"Images/runner-right3.gif");

rImages[4] =  getImage(getDocumentBase(),"Images/runner-left1.gif");

rImages[5] =  getImage(getDocumentBase(),"Images/runner-left2.gif");

rImages[6] =  getImage(getDocumentBase(),"Images/runner-left3.gif");


rImages[7] =  getImage(getDocumentBase(),"Images/rope-right1.gif");

rImages[8] =  getImage(getDocumentBase(),"Images/rope-right2.gif");

rImages[9] =  getImage(getDocumentBase(),"Images/rope-right3.gif");

rImages[10] =  getImage(getDocumentBase(),"Images/rope-left1.gif");

rImages[11] =  getImage(getDocumentBase(),"Images/rope-left2.gif");

rImages[12] =  getImage(getDocumentBase(),"Images/rope-left3.gif");

rImages[13] =  getImage(getDocumentBase(),"Images/stairs1.gif");
rImages[14] =  getImage(getDocumentBase(),"Images/stairs2.gif");


eImages = new Image[15];
eImages[0] =  getImage(getDocumentBase(),"Images/enemy0.gif");

eImages[1] =  getImage(getDocumentBase(),"Images/enemy-right1.gif");

eImages[2] =  getImage(getDocumentBase(),"Images/enemy-right2.gif");

eImages[3] =  getImage(getDocumentBase(),"Images/enemy-right3.gif");

eImages[4] =  getImage(getDocumentBase(),"Images/enemy-left1.gif");

eImages[5] =  getImage(getDocumentBase(),"Images/enemy-left2.gif");

eImages[6] =  getImage(getDocumentBase(),"Images/enemy-left3.gif");
```

```
    eImages[7] =  getImage(getDocumentBase(),"Images/enemy-rope-right1.gif");

    eImages[8] =  getImage(getDocumentBase(),"Images/enemy-rope-right2.gif");

    eImages[9] =  getImage(getDocumentBase(),"Images/enemy-rope-right3.gif");

    eImages[10] =  getImage(getDocumentBase(),"Images/enemy-rope-left1.gif");

    eImages[11] =  getImage(getDocumentBase(),"Images/enemy-rope-left2.gif");

    eImages[12] =  getImage(getDocumentBase(),"Images/enemy-rope-left3.gif");

    eImages[13] =  getImage(getDocumentBase(),"Images/enemy-stairs1.gif");
    eImages[14] =  getImage(getDocumentBase(),"Images/enemy-stairs2.gif");


    picture  = getImage(getDocumentBase(),"Images/picture.gif");
    lode  = getImage(getDocumentBase(),"Images/lode.gif");

    holeImages = new Image[6];
    holeImages[0] = getImage(getDocumentBase(),"Images/hole0.gif");

    holeImages[1] = getImage(getDocumentBase(),"Images/hole1.gif");

    holeImages[2] = getImage(getDocumentBase(),"Images/hole2.gif");

    holeImages[3] = getImage(getDocumentBase(),"Images/hole3.gif");

    holeImages[4] = getImage(getDocumentBase(),"Images/hole4.gif");
    holeImages[5] = getImage(getDocumentBase(),"Images/hole5.gif");


    for (int i = 0; i < holeImages.length; i++ ) mt.addImage(holeImages[i], 0);

    for (int i = 0; i < coreImages.length; i++ ) mt.addImage(coreImages[i], 0);

    for (int i = 0; i < rImages.length; i++ ) mt.addImage(rImages[i], 0);

    for (int i = 0; i < eImages.length; i++ ) mt.addImage(eImages[i], 0);

    mt.addImage(picture,0);
    mt.addImage(lode,0);

    try {
        mt.waitForAll();
    }
    catch (InterruptedException ex)   {
        System.err.println("Не могу загрузить изображения");
    }
}

public void init()  {
    //Вывод сообщения об инициализации
    initBuffer(this.getSize().width,this.getSize().height);
    this.setFont(font);
    this.setBackground (Color.black);
    buffer.getGraphics().fillRect (0, 0,
  this.getSize().width ,this.getSize().height);
    buffer.getGraphics().setColor(Color.yellow);
```

```
        this.setForeground(Color.yellow);
        String str = "Initializing ...";
        buffer.getGraphics().drawString (str,(this.getSize().width -
      this.getGraphics ().getFontMetrics(this.getFont()).stringWidth
      (str))/2,this.getSize().height/2);
        this.setForeground (Color.black);
        repaint();

        //Загрузка изображений
        loadImages();
        logger.setWritable( "true".equals(this.getParameter("log")));
        //Установка обработчика клавиатуры
        this.addKeyListener (this);
    }

    public void start() {
        A0(e00);
    }

//Загрузка уровня
    private void loadLevel() throws IOException {
        Color c = this.getForeground();
        Font f = this.getFont();
        this.setForeground  (Color.black);

        buffer.getGraphics().fillRect (0, 0,
        this.getSize().width ,this.getSize().height);
        this.setForeground(Color.yellow);
        this.setFont(font);
        String str = "Loading Level " + levelIndex + "...";
       buffer.getGraphics().drawString (str,(this.getSize().width -
      this.getGraphics ().getFontMetrics(this.getFont()).stringWidth
      (str))/2,this.getSize().height/2);
        this.setForeground  (Color.black);
        repaint();
        try {
            //level = new Level(this.getCodeBase(),coreImages,rImages);
            level.loadLevel("levels/level"+levelIndex+".lev");
            buffer.getGraphics ().dispose ();

            this.resize(level.getWidth () * Level.getCellSize (),level.getHeight ()
            * Level.getCellSize () + infoHeight);

            initBuffer(level.getWidth () * Level.getCellSize (),level.getHeight ()
            * Level.getCellSize () + infoHeight);
            buffer.getGraphics ().setXORMode(Color.black);
            sbuffer = createImage(buffer.getWidth(null)  , buffer.getHeight(null));
            level.paint(sbuffer.getGraphics ());
            buffer.getGraphics().drawImage(sbuffer,0,0,null);
            men = level.getMen ();
            for (int i = 0; i< men.length; i++ ) men[i].draw(buffer.getGraphics());
```

```
            // Таймер
            timer = new Timer();
            timer.schedule(new TimerTask () {   public void run() {
                                                                    A0(e10);
                                                                   }

                                                                 }, 0,55);
            repaint();
        } catch (IOException ex) {
            throw ex;
        }
        repaint();
        this.setForeground(c);
        this.setFont(f);
    }


//-------------------------------------------------------------------------------
//---Входные переменные автомата A0-----------------------------------------------
//-------------------------------------------------------------------------------

// Герой в центре клетки?
    private boolean x00(){
        if ((men[0].getPos().getInX() == 0)&& (men[0].getPos().getInY() == 0))
return true;
        else return false;
    }

// Герой на выходе с уровня?
    private boolean x01(){
        if (level.isExit(men[0].getPos().getY(),men[0].getPos().getX()))
return true;
        else return false;
}

// Загружен ли  новый уровень?
    private boolean  x02(){
        timer.cancel();
        levelIndex++;
        try {
            loadLevel();
        } catch (IOException ex){
            return false;
        }
        return true;
    }

// Герой мертв?
    private boolean x03(){
        for (int i = 1; i < men.length; i++) {
            if (((Enemy)men[i]).isKill()) return true;
        }
        return ((Runner) men[0]).inHole();
    }

// Есть жизни?
    private boolean x04(){
        if (((Runner) men[0]).getLives() > 1) return true;
            return false;    }
```

```
//-----------------------------------------------------------------------------
//--Выходные воздействия автомата A0-----------------------------------------
//-----------------------------------------------------------------------------

// Создание объекта Level
    private void z00() {
        logger.log(1, "z00 - создание объекта Level");
        level = new Level(this.getCodeBase(),coreImages,rImages, eImages);
    }

// Загрузка уровня и запуск таймера
    private void z01(){
        logger.log(1, "z01 - загрузка уровня и запуск таймера");
        try{
            loadLevel();
        } catch (IOException ex) {
            System.out.println (ex);
            System.exit(1);
        }
    }

// Вывод игрового пространства
    private void z02_1() {
        logger.log(1, "z02_1 - вывод игрового пространства");
        buffer.getGraphics().drawImage (sbuffer,0,0,null);
        buffer.getGraphics().setPaintMode();
        buffer.getGraphics().setXORMode(Color.black);

        if (level.lodes.size() == 0) { level.showExit();
      level.paint(buffer.getGraphics());}

        for (int i  = level.holes.size()-1; i>=0 ; i--){

            if (((Hole)level.holes.get(i)).isDead) level.holes.remove(i);
        }

        level.paintHoles(buffer.getGraphics());
        level.paintLodes(buffer.getGraphics());

        men[0].draw (buffer.getGraphics());
        for (int i  = 1; i < men.length; i++){
            men[i].draw (buffer.getGraphics());
        }
        String str = "level:" + levelIndex + "   boxes: " + level.lodes.size() + "
      lives: " + ((Runner) men[0]).getLives();
        buffer.getGraphics().drawString (str, 2, -2 + buffer.getHeight(null));
        paint(this.getGraphics ());
    }

// Вывод заставки
    private void z02_2(){
        logger.log(1, "z02_2 - вывод заставки");
        this.setForeground(Color.black);
        buffer.getGraphics().fillRect (0, 0,
      this.getSize().width ,this.getSize().height);
        buffer.getGraphics().drawImage (picture,0,0,null);
        this.setForeground  (Color.red);
        this.setFont(new Font("Serif", Font.ITALIC, 15));
        String str1 = "press \"space\" to start";
```

```
        buffer.getGraphics().drawString(str1, ( -
      this.getFontMetrics(this.getFont()).stringWidth(str1) +
      buffer.getWidth(null)) / 2 ,buffer.getHeight(null)/2 + 2);
        this.setFont(new Font("Serif", Font.ITALIC, 12));
        String str = "left - left arrow  right - right arrow  up - up arrow  down -
      down arrow";
        String str2 = "(c) Bogdanov Michael, IFMO 2004, flown@bk.ru";
        buffer.getGraphics().drawString(str, ( -
      this.getFontMetrics(this.getFont()).stringWidth(str) + buffer.getWidth(null))
      / 2 ,buffer.getHeight(null) - this.getFontMetrics(this.getFont()).getHeight()
      - 10);

      buffer.getGraphics().drawString(str2, ( -
      this.getFontMetrics(this.getFont()).stringWidth(str2) +
      buffer.getWidth(null)) / 2 ,buffer.getHeight(null) - 10);
        this.setForeground(Color.black);
        repaint();
    }

//Вывод результатов
    private void z02_3() {
        logger.log(1, "z02_3 - вывод результатов");
        this.setFont(font);
        this.setForeground(Color.black);
        buffer.getGraphics().fillRect (0, 0,
      this.getSize().width ,this.getSize().height);
        this.setForeground(Color.yellow);
        String str = "Game over!!!";
        buffer.getGraphics().drawString (str,(this.getSize().width -
      this.getGraphics ().getFontMetrics(this.getFont()).stringWidth
      (str))/2,this.getSize().height/2);

        this.setForeground  (Color.red);
        this.setFont(new Font("Serif", Font.ITALIC, 15));
        String str1 = "press \"space\" to continue";
        buffer.getGraphics().drawString(str1, ( -
      this.getFontMetrics(this.getFont()).stringWidth(str1) +
      buffer.getWidth(null)) / 2 ,buffer.getHeight(null) - 10);
        this.setForeground(Color.black);
        repaint();
    }

// Движение врагов
    private static int counter = 15;
    private void z03() {
        logger.log(1, "z03 - движение врагов");
        counter--;
        if (counter < 0 ) { counter = 15; level.buildRoutes(men[0].getPos().getX(),
      men[0].getPos().getY()); }
        for (int i = 1; i < men.length; i++)
            ((Enemy)men[i]).move();
    }


// Обновление состояния золота
    private void z04(){
        logger.log(1, "z04 - обновление состояния золота");
      if ((men[0].getPos().getInX() == 0)&& (men[0].getPos().getInY() == 0)) {
            for (int i = level.lodes.size()-1; i >= 0 ; i--){
```

```
      if(((Lode)level.lodes.get(i)).isEqual(men[0].getPos().getX(),men[0].getPos().
      getY())){
                    ((Runner)men[0]).addLode();
                    level.lodes.remove(i);
              }
              }
      }
    }

// Вывод сообщения о гибели героя
    private void z05(){
        logger.log(1, "z05 - вывод сообщения о гибели героя");
        isWaiting = true;
        try {
            String str1 = "Ooopps...";
            this.setFont(new Font("Serif", Font.ITALIC, 24));
            this.setForeground(Color.red);
            buffer.getGraphics().drawString(str1, ( -
            this.getFontMetrics(this.getFont()).stringWidth(str1) +
            buffer.getWidth(null)) / 2 ,buffer.getHeight(null)/2 + 2);
            repaint();
            Thread.currentThread().sleep(1500);
            z09();
        } catch(Exception t) {
            System.out.println("error " + t);
        }
    }

// Отключение таймера
    private void z06(){
        logger.log(1, "z06 - отключение таймера");
        timer.cancel();
    }

// Установка первого уровня
    private void z07(){
        logger.log(1, "z07 - установка первого уровня");
        levelIndex = 1;
    }
// Снятие жизни
    private void z08(){
        logger.log(1, "z08 - снятие жизни");
        ((Runner)men[0]).subLive();
        men[0].resetAutomat();
    }

// Настройка шрифтов
    private void z09() {
        logger.log(1, "z09 - настройка шрифтов");
        this.setFont(new Font("Serif", Font.BOLD, 10));
        this.setForeground(Color.red);
    }

// Перезагрузка уровня
    private void z10() {
        logger.log(1, "z10 - перезагрузка уровеня");
        level.reInitLevel();
        men = level.getMen();
        timer = new Timer();
```

```
        timer.schedule(new TimerTask () {
                                public void run() {
                                    A0(e10);
                                }
                            }, 0,55);
        isWaiting = false;
        keyPressed = e00;
    }

//-------------------------------------------------------------------------------
//--Автомат A0-взаимодействие с игроком-----------------------------------------
//-------------------------------------------------------------------------------
private  void A0(final int e) {
        int oldState = state;
        logger.log("Автомат A0 запущен с событием ", e);
    switch (state) {
            case y00:
                state = y01;
            break;

            case y01:
                if (e == e05){
                    z01();
                    state = y02;
                }
            break;

            case y02:  //игра
                if (e == e10 && x00() && x01() && !x02()) { //конец
                    state = y03;
                    break;
                }

                if (e == e10 && x03() && !x04()) {
                    z06();
                    state = y03;
                    break;
                }
                if (e == e10 && x03() && x04()) {
                    z06();
                    z05();
                    z08();
                    z10();
                    z02_1();
                    break;
                }
                if (e == e10 && !x03()) {
                    men[0].A1(keyPressed);
                    z03();
                    z04();
                    z02_1();
                    break;
                }

                if (e == e06) {    //нажата клавиша escape
                    z06();
                    state = y01;
                }
            break;
```

```
            case y03:   //конец
                  if (e05 == e) {
                    state = y01;
                }
            break;
        }

        if (oldState != state) {
            logger.log("Автомат A0 перешел из состояния y0" + oldState + " в y0" +
            state);

            switch (state){
                  case y00:
                break;

                case y01:
                    z00();
                    z07();
                    z02_2();
                break;

                case y02:
                  z09();
                break;

                case y03:
                    z02_3();
                break;
            }
        }
        logger.log("Автомат A0 завершил обработку события ", e);
    }


// Вывод изоюражения
    public  void paint(Graphics g){
        g.drawImage(buffer, 0, 0, null);
    }


// Интерфейс KeyListner

    public void keyPressed(KeyEvent e) {
        if (!isWaiting)
        switch (e.getKeyCode()) {
            case KeyEvent.VK_UP:
                keyPressed = e01;
                break;
            case KeyEvent.VK_DOWN:
                keyPressed = e02;
                break;
            case KeyEvent.VK_LEFT:
                keyPressed = e03;
                break;
            case KeyEvent.VK_RIGHT:
                keyPressed = e04;
                break;
            case KeyEvent.VK_SPACE:
```

```
                keyPressed = e05;
                if (state == y01) A0(e05);
                break;
            case KeyEvent.VK_ESCAPE :
                keyPressed = e06;
                if (state == y02) A0(e06);
                break;
            default:
                keyPressed = e00;
            }
    }

    public void keyReleased(KeyEvent e) {
        int key = 0;
        if (!isWaiting) {
            switch (e.getKeyCode()) {
                case KeyEvent.VK_UP:
                    key = e01;
                    break;
                case KeyEvent.VK_DOWN:
                    key = e02;
                    break;
                case KeyEvent.VK_LEFT:
                    key = e03;
                    break;
                case KeyEvent.VK_RIGHT:
                    key = e04;
                    break;
                case KeyEvent.VK_SPACE:
                    key = e05;
                    if (state != y02) A0(e05);
                    break;
                case KeyEvent.VK_ESCAPE :
                    key = e06;
                    if (state == y02) A0(e06);
                    break;
                default:
                    key = e00;
            }
           if (keyPressed == key) keyPressed = e00;
        }
    }

    public void keyTyped(KeyEvent e) {}


    public void destroy(){
      A0(e06);
    }
}
```

2. Класс *Level*

```
package loderunner;

import java.io.*;
import java.net.*;
import java.awt.*;
import java.util.*;

class LevelFormatException extends IOException {}
/*
 * Осуществляет загрузку карты и предоставляет доступ к ней.
 */
public class Level {

    //Элементы карты
    static int EMPTY = 0;
    static int BRICK1 = 1;
    static int BRICK2 = 2;
    static int STAIRS = 3;
    static int EXITSTAIRS = 5;
    static int ROW = 4;
    static int LODE = 6;
    private static int RUNNER = 7;
    private static int ENEMY = 8;
    private static int TRAP = 9;
    static int HOLE = 10;
    static int MANINHOLE = 11;

    //Параметры карты
    private static int width = 32;
    private static int height = 22;
    private static int cellSize = 16;
    private static int [] coreCodes = {1,2,3,4};
    private byte [][] map, sourceMap, grayMap;
    private final URL fileBase;
    private int boxesNumber, enemysNumber;
    Runner runner;
    Enemy [] enemys;
    private Vector exitStairs;
    Vector enemyList;
    public Vector holes;
    public Vector lodes;
    private Vector traps;
    Image coreImages[];
    Image rImages[], eImages[];
    private ArrayList gray;
    private byte grayINF = 127;


    Level(URL fileBase,Image images[],Image [] rImages, Image [] eImages){
        this.fileBase = fileBase;
        map = new byte [height] [width];
        sourceMap = new byte [height] [width];
        grayMap = new byte [height] [width];

        coreImages = images;
        this.rImages = rImages;
        this.eImages = eImages;
```

```
    holes = new Vector ();
    lodes = new Vector();
    exitStairs = new Vector();
    traps = new Vector();
}

// Загрузка уровня
void loadLevel(String fileName) throws IOException,LevelFormatException {

    URL url = new URL(fileBase + fileName);
    URLConnection urlConnection = url.openConnection();
    InputStream iStream = urlConnection.getInputStream();
    BufferedReader bFile = new BufferedReader(new InputStreamReader(iStream));
    try {
        String str;
        for (int i =0; i<height; i++){
            str = bFile.readLine();
            if ((str ==  null) || (str.length () != width )) throw new
        LevelFormatException();
            for (int j = 0; j < width;  j++)
            sourceMap[i][j] = Byte.parseByte(str.substring (j,j+1));
        }
    } catch (IOException e) {
        if (bFile != null) bFile.close();
        if (iStream != null) iStream.close();
        throw e;
    }
    reInitLevel();
}

// Возвращение уровня к начальному состоянию
public void reInitLevel(){
    lodes.clear();
    holes.clear();
    exitStairs.clear();
    traps.clear();
    enemysNumber = 0;
    boxesNumber = 0;

    for (int i =0; i<height; i++)
        for (int j = 0; j < width;  j++)
            map[i][j] = sourceMap[i][j];
    for (int i =0; i<height; i++)
        for (int j = 0; j < width;  j++) {
            boxesNumber += (map[i][j] == LODE) ? 1 : 0;
            if (map[i][j] == LODE) {
                lodes.add(new Lode(j,i));
                map[i][j] = (byte)EMPTY;
            }
            if (map[i][j] == TRAP) { traps.add(new Byte ((byte)i));
        traps.add(new Byte((byte)j)); map[i][j] = (byte)EMPTY;}
            enemysNumber += (map[i][j] == ENEMY) ? 1 : 0;
            if (map[i][j] == EXITSTAIRS) {
                exitStairs.add(new Byte((byte)i));
                exitStairs.add(new Byte((byte)j));
                map[i][j] = (byte)EMPTY;
            }
        }
    try {
```

```
        createMans();
        buildRoutes(runner.getPos().getX(),runner.getPos().getY());
    } catch (LevelFormatException e){
    }

    try {
        Thread.currentThread().wait();
    } catch (Exception e) {
    }
}

//Создание Персонажей
private void createMans() throws LevelFormatException {
    enemyList = new Vector ();
    for (int i = 0; i < height; i++)
        for (int j = 0; j < width; j++) {
            if (map[i][j] == ENEMY)   {
                enemyList.addElement(new Enemy (eImages,this,j,i));
                map[i][j] = (byte)EMPTY;
            }

            if (map[i][j] == RUNNER)
                if (runner == null)
                    runner = new Runner (rImages,this,j,i);
                else runner.setPos(new Position(j,i));
        }

    if (runner == null) throw new LevelFormatException();
    map[runner.pos.getY()][runner.pos.getX()] = (byte)EMPTY;

    for (int i = 0; i < enemyList.size(); i++)
        ((Enemy) enemyList.get(i)).setHeroe(runner);
}

//Возвращает список всех персонажей
public Man [] getMen(){
    Man [] men = new Man [1+enemyList.size()];
    men[0] = runner;
    for (int i = 0; i < enemyList.size();i++)
        men[i+1] = (Enemy) enemyList.elementAt(i);
    return men;
}

//Прорисовка уровня
public void paint(Graphics g){
    for (int i = 0; i < height; i++)
        for (int j = 0; j < width; j++)
            if ((map[i][j] >= 1) && (map[i][j] <= 4))
              g.drawImage(coreImages [map[i][j]-1], cellSize*j, cellSize*i,
              null);
    for (int i = 0; i < traps.size(); i += 2) {
        g.drawImage(coreImages [0], cellSize * ((Byte)traps.get(i +
        1)).byteValue(), cellSize * ((Byte)traps.get(i)).byteValue(), null);
    }
}

//Прорисовка ям
public void paintHoles(Graphics g){
    for (int i = 0; i < holes.size(); i++)
```

```
        ((Hole)holes.get(i)).draw(g);
}

//Прорисовка золота
public void paintLodes(Graphics g){
    for (int i = 0; i < lodes.size(); i++)
        ((Lode)lodes.get(i)).draw(g);
}

//Возвращает число слитков
public int getBoxesNumber() {return boxesNumber; }

//Возвращает количество врагов
public int getEnemysNumber() {return enemysNumber; }

//Возвращает размер ячейки
public static int getCellSize(){
    return cellSize;
}

//Возвращает ширину экрана в ячейках
public int getWidth(){
    return width;
}

//Возвращает высоту экрана в ячейках
public int getHeight(){
    return height;
}

//Возвращает тип ячейки в позиции (x,y) с игровой карты
public int getBoxAT(int y, int x){
    if ((x>=width) || (x<0) || (y<0) || (y >= height)) return BRICK2;
    else return map[y][x];
}

//Возвращает тип ячейки в позиции (x,y) из файла, описывающего уровень
public int getSBoxAT(int y, int x){
    if ((x>=width) || (x<0) || (y<0) || (y >= height)) return BRICK2;
    else if ((sourceMap[y][x] != LODE) && (sourceMap[y][x] != RUNNER) &&
  (sourceMap[y][x] != ENEMY) && (sourceMap[y][x] != EXITSTAIRS)) return
  sourceMap[y][x];
        else return map[y][x];
}

//Возвращает эвристиику, соответствующую ячейке (x,y), используется для
управления врагами
public byte getEvrAT(int y, int x){
    if ((x>=width) || (x<0) || (y<0) || (y >= height)) return grayINF;
    else return grayMap[y][x];
}

//Создает новую яму
public void newHole(int x, int y){
    boolean f =false;
    for (int i =0; i < holes.size(); i++) if
  (((Hole)holes.get(i)).isEqual(x,y)) {f = true; break;}
    if (!f)   holes.add(new Hole(x,y,this));
}
```

```
  //Меняет тип ячейки (x,y)
public void setBoxAT(int y, int x, int  type){
    map[y][x] = (byte)type;
}

//Выводит игровую лестницу
public void showExit(){
    for (int i=0; i < exitStairs.size(); i=i+2){

        map[((Byte)exitStairs.get(i)).byteValue()][((Byte)exitStairs.get(i+1)).
        byteValue()] = (byte)STAIRS;
    }
}

//Определяет находится ли герой на выходе с уровня
public boolean isExit(int y, int x){
    int l = exitStairs.size ();
    if ((y == 0) && (getBoxAT(y,x) == STAIRS)) return true;
    else return false;
}

//Заполняет массив эвристик, который используется для управления врагами
public void buildRoutes(int x, int y) {
    gray = new ArrayList();
    gray.add(new Integer(y * 32 + x));
    for (int i = 0;  i < height; i++)
        for (int j = 0; j < width; j++)
            grayMap[i][j] = grayINF;

    grayMap[y][x] = 0;
    while (gray.size() > 0) {
        Object o = gray.remove(0);
        if (o != null) {
            y = ((Integer) o).intValue() / 32;
            x = ((Integer) o).intValue() % 32;

            if ((getSBoxAT(y - 1, x) != BRICK1) && (getSBoxAT(y - 1, x) !=
            BRICK2) && (grayMap[y - 1][x] == grayINF)) {
                gray.add(new Integer( (y - 1) * 32 + x));
                grayMap[y - 1][x] = (byte)(grayMap[y][x] + 1);
            }

            if  (((getSBoxAT(y, x + 1) == STAIRS) || (getSBoxAT(y, x + 1) ==
        ROW) || (((getSBoxAT(y + 1, x + 1) == BRICK1) || (getSBoxAT(y + 1, x +
        1) == BRICK2)||(getSBoxAT(y + 1, x + 1) == STAIRS)) && (getSBoxAT(y, x
        + 1) == EMPTY)) ) && (grayMap[y][x + 1] == grayINF)) {
                gray.add(new Integer( y * 32 + x + 1));
                grayMap[y][x + 1] = (byte)(grayMap[y][x] + 1);
            }

            if ((getSBoxAT(y + 1, x) == STAIRS) && (grayMap[y + 1][x] ==
            grayINF)) {
                gray.add(new Integer( (y + 1) * 32 + x));
                grayMap[y + 1][x] = (byte)(grayMap[y][x] + 1);
            }

            if  (((getSBoxAT(y, x - 1) == STAIRS) || (getSBoxAT(y, x - 1) ==
            ROW) || (((getSBoxAT(y + 1, x - 1) == BRICK1) || (getSBoxAT(y + 1,
```

```
                x - 1) == BRICK2) || (getSBoxAT(y + 1, x - 1) == STAIRS)) &&
                (getSBoxAT(y, x - 1) == EMPTY))) && (grayMap[y][x - 1] == grayINF))
                {
                    gray.add(new Integer( y * 32 + x - 1));
                    grayMap[y][x - 1] = (byte)(grayMap[y][x] + 1);
                }
            }
        }
    }

    // возвращает псевдобесконечность;
    public byte getINF() {
        return grayINF;
    }
}
```

3. Класс *Enemy*

```
package loderunner;

import java.awt.Image;
import java.util.Random;

/*
 *класс Enemy - управление врагом
 */
public class Enemy extends Man {

    private Man heroe;
    private static Random random;
    private Lode lode = null;
    private Hole hole = null;
    private int inHole = 0;

    Enemy(Image [] images,Level level,int x,int y){
        super(images,level,x,y);
        direction = dirLEFT;
        random = new Random();
        lode = null;
    }

// Привязка героя
    public void setHeroe(Man h) {
        heroe = h;
    }

// Убит ли герой данным enemy?
    public boolean isKill(){
        Position hp = heroe.getPos();
        Position ep = getPos();
        if ( (Math.abs( 3 * (hp.getX() - ep.getX()) + hp.getInX() - ep.getInX()) <
      3)  &&  (Math.abs( 3 * (hp.getY() - ep.getY()) + hp.getInY() - ep.getInY() )
      < 3) ) return true;
        return false;
    }

// Возвращает направление движения
    private int getDirection() {
```

```
        int dir = dirSTAY;
        byte min = level.getINF();
        int x = pos.getX();
        int y = pos.getY();
        if (min >= level.getEvrAT(y - 1, x)) {min = level.getEvrAT(y - 1, x); dir =
      dirUP; }
        if (min >= level.getEvrAT(y + 1, x)) {min = level.getEvrAT(y + 1 , x); dir
      = dirDOWN; }
        if (min >= level.getEvrAT(y, x + 1)) {min = level.getEvrAT(y, x + 1); dir =
      dirRIGHT; }
        if (min >= level.getEvrAT(y, x - 1)) {min = level.getEvrAT(y, x - 1); dir =
      dirLEFT; }
        return dir;
    }

// Движение врага
    public void move() {
        //смерть в яме
        if (inHole()) {
            pos = new Position (random.nextInt(level.getWidth()), 0);
            resetAutomat();
            inHole = 0;
            hole = null;
        }

        //отъямивание
        if (inHole > 0) {
            if ((hole != null) && (!hole.isClosing())) {
                if (inHole == 5) level.setBoxAT(pos.getY(), pos.getX(),Level.HOLE);
                if ((inHole >= 4) && (inHole <= 6)) pos.addInY(-1);
                if (inHole == 4) {
                    direction = getDirection();
                    if ((direction != dirLEFT) && (direction != dirRIGHT)) {
                        direction = dirSTAY;
                    }
                }
                if ((inHole >= 1) && (inHole <= 3)) step(direction);
            } else { inHole = 1;}
            inHole--;
            if (inHole == 0) hole = null;
        } else {
         if ((pos.getInX() == 0) && (pos.getInY() == 0) && (getCellType(Z) ==
      Level.HOLE)) {
                level.setBoxAT(pos.getY(), pos.getX(), Level.BRICK2);
                putLode();
                inHole = 40;
                hole = Hole.getHole(pos);
            } else if (!((pos.getInX() == 0) && (pos.getInY() == 0) &&
            (getCellType(Z) == Level.BRICK2))){
                if ((pos.getInX() == 0) && (pos.getInY() == 0)) { takeLode();
                direction = getDirection(); }
                A1(direction);
            }
        }
    }

//В зависимости от наличия золота в текущей ячейке, решает брать его или нет
    private void takeLode() {
        if (lode == null)
```

```
            for (int i = 0; i < level.lodes.size(); i++) {
                if (( (Lode)level.lodes.get(i)).isEqual(pos.getX(),pos.getY()) ) {
                    if (random.nextInt(10) <= 2) {
                        lode = (Lode)level.lodes.get(i);
                        lode.setStatic(false);
                        break;
                    }
                }

            }
    }

 //Выбрасывает золото на уровень
   private void putLode() {
       if (lode != null) {
           lode.setPosition(pos.getX(), pos.getY() - 1);
           lode.setStatic(true);
           lode = null;
       }
   }
}
```

4. Класс *Runner*

```
package loderunner;

import java.awt.*;
/*
 * Класс Runner - управление героем.
 */
public class Runner extends Man {

    private int score;
    private int lodeNumber;
    private int lives;

    Runner (Image [] rImages,Level  level,int x,int y){
      super(rImages,level,x,y);
      lodeNumber  = 0;
        lives = 3;
    }

// Увеличить количество собранного золота на 1
    public void  addLode(){
        lodeNumber++;
    }

//Возвращает количество жизней
    public int getLives() {
        return lives;
    }

 //Отнимает одну жизнь
    public void subLive() {
        lives--;
    }
}
```

5. Классы *Man* и *Position*

```
package loderunner;

import java.awt.*;

/*
 * Хранит позицию игрока
 */
class Position {

    private int x,y;
    private int inX,inY;

    public Position(int x, int y){
        this.x = x;
      this.y = y;
      inX = 0;
        inY = 0;
    }

    public int getX(){
        return x;
    }

    public int getY(){
        return y;
    }
    public int getInX(){
        return inX;
    }
    public int getInY(){
        return inY;
    }

    public void addInX(int x){
        this.inX += x;
      if (inX == 2) { this.x++; inX = -1 ;}
      else if (inX == -2) {this.x--; inX = 1;}
    }

    public void addInY(int y){
        this.inY += y;
        if (inY == 2) { this.y++; inY = -1 ;}
        else if (inY == -2) {this.y--; inY = 1;}
    }
}

/*
 * Обрабатывает управляющие сигналы и осуществляет передвижение героя и врагов.
 */
public class Man {

    protected final int Z = 0;
    protected final int W = 1;
    protected final int E = 2;
    protected final int S = 3;
    protected final int N = 4;
    protected final int SW = 5;
```

```
    protected final int SE = 6;
    protected final int Sn = 7;
    protected final int Nn = 8;
    protected final int Zn = 9;

    //состояния автомата A1
    private final int y11 = 11; //на поверхности
    private final int y12 = 12; //на веревке
    private final int y13 = 13; //на лестнице
    private final int y14 = 14;
    private int state = y11;

    //направления движения
    final static int dirSTAY = 0;
    final static int dirUP = 1;
    final static int dirDOWN = 2;
    final static int dirLEFT = 3;
    final static int dirRIGHT = 4;
    protected int direction = dirSTAY;

    protected Image [] images;
    private int pstate;
    protected int imageState;

    protected Position pos;
    final static int cellSize = 16;
    protected Level level;

    protected int   po=-1;

    public Man(Image [] images,Level level,int x,int y){
        this.images = images;
        pos = new Position (x,y);
        imageState = 0;
        pstate = 0;
        this.level = level;
    }

    //Прорисовка персонажа
    public void draw(Graphics g){
        g.drawImage(images[imageState],cellSize*pos.getX() +
Math.round(pos.getInX()*(float)5.5),cellSize*pos.getY() +
Math.round(pos.getInY()*(float)5.5),null);
    }

    //Шаг в горизонтальном направлении
    public void step(int dir){

          int sign = 1;
        if (dir != dirRIGHT) sign = -1;

        if (pstate == 0) pstate = 2 * sign;
        else pstate = sign * (Math.abs(pstate) % 3 + 1);

        if (pstate > 0) {
            pos.addInX(1);
            imageState = pstate;
        } else {
            pos.addInX(-1);
```

```
            imageState = 3 + Math.abs(pstate);
        }
    }

      //Возвращает позицию персонажа
    public Position getPos(){
        return pos;
    }

    //Расстояние между двумя Man'ами
    protected int dest(Position a, Position b){
      if ((Math.abs(3 * a.getX() + a.getInX() - 3 * b.getX() - b.getInX()) < 3) &&
      (Math.abs(3 * a.getY() + a.getInY() - 3 * b.getY() - b.getInY()) < 3)) return
      0;
        return 1;
    }

    // Находится ли Man в яме?
    protected boolean inHole(){
        int y = this.getPos().getY();
        int x = this.getPos().getX();
        for (int i = -1; i <= 1; i++) {
            for (int j = -1; j <= 1; j++) {
            if ( level.getBoxAT(y + j, x + i) == Level.BRICK1)
                 if (dest(this.getPos(), new Position (getPos().getX() + i,
            getPos().getY() + j)) < 1){ return true;}
            }
        }
        return false;
    }

    //Устанавливает позицию персонажа
    public void setPos(Position p) {
        pos = p;
    }

    //Возвращает тип ячейки относительно позиции персонажа
    protected int getCellType(int c){
        switch (c) {
            case Z:  return level.getBoxAT(pos.getY(), pos.getX());
            case E:  return level.getBoxAT(pos.getY(), pos.getX() + 1);
            case W:  return level.getBoxAT(pos.getY(), pos.getX() - 1);
            case S:  return level.getBoxAT(pos.getY() + 1, pos.getX() );
            case N:  return level.getBoxAT(pos.getY() - 1, pos.getX() );
            case SE: return level.getBoxAT(pos.getY() + 1, pos.getX() + 1);
            case SW: return level.getBoxAT(pos.getY() + 1, pos.getX() - 1);
            case Sn: return level.getBoxAT(pos.getY()+1,pos.getX()+ pos.getInX());
            case Zn: return level.getBoxAT(pos.getY(), pos.getX() + pos.getInX());
            default : return Level.EMPTY;
        }
    }

//-------------------------------------------------------------------------------
//---Входные переменные автомата A1-----------------------------------------------
//-------------------------------------------------------------------------------

// Движется влево?
    protected boolean xl(){
        if (direction == dirLEFT) return true;
```

```
        else return false;
    }

// Движется вправо?
    protected boolean xr(){
        if (direction == dirRIGHT) return true;
        else return false;
    }

// Далее ячейка берется относительно позиции игрока
// Ячейка пуста (EMPTY)?
    protected boolean xc0(int c){
        if ((getCellType(c) == Level.EMPTY) || (getCellType(c) == Level.HOLE))
return true;
        else return false;

    }
// Ячейка BRICK1?
    protected boolean xc1(int c){
        if (getCellType(c) == Level.BRICK1) return true;
        else return false;
    }

// Ячейка BRICK2?
    protected boolean xc2(int c){
        if (getCellType(c) == Level.BRICK2) return true;
        else return false;
    }

// Ячейка STAIRS?
    protected boolean xc3(int c){
        if (getCellType(c) == Level.STAIRS) return true;
        else return false;
    }

// Ячейка ROW?
    protected boolean xc4(int c){
        if (getCellType(c) == Level.ROW) return true;
        else return false;
    }

// Позиция внутри ячейки по горизонтали: слева, справа, в центре
    protected boolean xp(int p) {
        switch (p) {
            case Z: if (pos.getInX() == 0 ) return true; break;
            case W: if (pos.getInX() == -1) return true; break;
            case E: if (pos.getInX() == 1) return true; break;
        }
        return false;
    }

// Позиция внутри ячейки по вертикали: сверху, снизу, в центре
    protected boolean xpy(int p) {
        switch (p) {
            case S: if (pos.getInY() == 1) return true;
            case Z: if (pos.getInY() == 0) return true;
        }
        return false;
    }
```

```
//-------------------------------------------------------------------------------
//--Выходные воздействия автомата A1-------------------------------------------
//-------------------------------------------------------------------------------

// Шаг вверх
    protected void zu() {
        Game.logger.log(2,"zu - шаг вверх");
        po++; pos.addInY(+1); imageState = 13 + po;
        if (1==po) po=-1;
    }

// Шаг вниз
    protected void zd() {
        Game.logger.log(2,"zu - шаг вниз");
        po++; pos.addInY(-1); imageState = 13 + po;
        if (1==po) po=-1;
    }

// Элементарный шаг по y
    protected void zy (int d) {
        Game.logger.log(2,"zy - элементарный шаг по y");
        switch (d) {
            case S: pos.addInY(+1); direction = dirDOWN; break;
            case N: pos.addInY(-1); direction = dirUP; break;
        }
    }

// Нужно для состояния 3
    protected void z03() {
        Game.logger.log(2,"z03 - нужно для состояния 3");
        po++;
        imageState = 13 + po;
        if (1==po) po=-1;
    }

// Один цикл падения
    protected void zf() {
        Game.logger.log(2,"zf - один шаг полета");
        imageState = 0;
        pos.addInY(+1);
    }

// Шаг по горизонтали
    protected void zs(int d){
        Game.logger.log(2,"zs - шаг по горизонтали");
        int p = dirSTAY;
        if (d == W) p = dirLEFT;
        else if (d == E) p = dirRIGHT;
        step(p);
        direction  = p;
    }

// Выкопать яму
    private void zh(int j) {
        Game.logger.log(2,"zh - выкопать яму");
        switch (j) {
            case SE:
                level.newHole (pos.getY()+1, pos.getX() + 1);
```

```
                break;
            case S:
                level.newHole (pos.getY()+1, pos.getX());
                break;
            case SW:
                level.newHole (pos.getY()+1, pos.getX() - 1);
                break;
        }
    }

//Сменить картинку персонажа
    private void zi(int i) {
        if (i < 0) imageState = 0;
        else imageState += i;
    }


//-------------------------------------------------------------------------------
//-- Автомат: управление персонажем-----------------------------------------------
//-------------------------------------------------------------------------------
    public void A1(final int e){
        Game.logger.log(1,"Автомат A1 запущен с событием ", e);
        int oldstate  = state;
        switch (state) {
            case y11:  //на кирпичах
                if (inHole()) { state = y11; break;}
                if (xp(Z) && (xc0(S) || xc4(S)) && !xc3(Z)) { //Fly //****
                    state = y14;
                    break;
                }

                if ((xc0(S) || xc4(S)) && (xc0(Sn) || xc4(Sn)) && !(xc3(Z) ||
                xc3(Zn))) { //Fly //****
                    state = y14;
                    break;
                }

                if (e == Game.e05 && xl() && !xp(E) &&  xc1(SW) && !xc1(W)
                && !xc2(W)) {
                    zh(SW);
                    break;
                }

                if (e == Game.e05 && xl() && xp(E) && xc1(S)) {
                    zh(S);
                    break;
                }

                if (e == Game.e05 && xr() && !xp(W) &&  xc1(SE) && !xc1(E)
                && !xc2(E)) {
                    zh(SE);
                    break;
                }

                if (e == Game.e05 && xr() && xp(W) && xc1(S)) {
                    zh(S);
                    break;
                }
```

```
        if (e == Game.e03 && xp(E)) {
            zs(W);
            break;
        }

        if (e == Game.e03 && !xp(E) && !xc1(W) && !xc2(W)) {
            zs(W);
            break;
        }

        if (e == Game.e04 && xp(W)) {
            zs(E);
            break;
        }

        if (e == Game.e04 && !xp(W) && !xc1(E) && !xc2(E)) {
            zs(E);
            break;
        }

        if (e == Game.e01 && xp(Z) && !xc1(N) && !xc2(N) && xc3(Z)) {
            zy(N);
            state = y13;
            break;
        }

        if (e == Game.e02 && xp(Z) && (xc3(S) || xc0(S))) {
            zy(S);
            state = y13;
            break;
        }
    break;

    case y12:   //на веревке
        if (e == Game.e02 && xc0(S) && xc0(Sn)) {
            zy(S);
            state = y14;
            break;
        }
        if (e == Game.e02 &&  !(xc0(S) && xc0(Sn))) {
            state = y11;
            break;
        }

        if (xp(Z) && !xc4(Z) && (xc1(S) || xc2(S) || xc3(S) || xc3(Z))) {
            state = y11;
            break;
        }

        if (xp(Z) && !xc4(Z) && !(xc1(S) || xc2(S) || xc3(S) || xc3(Z))) {
            state = y14;
            break;
        }

        if (e == Game.e03 && xp(E)) {
            zs(W);
            zi(6);
            break;
        }
```

```
        if (e == Game.e03 && !xp(E) && !xc1(W) && !xc2(W)) {
            zs(W);
            zi(6);
            break;
        }

        if (e == Game.e04 && xp(W)) {
            zs(E);
            zi(6);
            break;
        }

        if (e == Game.e04 && !xp(W) && !xc1(E) && !xc2(E)) {
            zs(E);
            zi(6);
            break;
        }
    break;

    case y13:   //на лестнице
        if (e == Game.e01 && (xpy(S) || (!xc1(N) && !xc2(N)))) zd();
        if (e == Game.e02) {zu();}
        if (xpy(Z)) state = y11; //****
    break;

    case y14:   //паденение
          if (xpy(Z) &&(xc1(S) || xc2(S)  || xc3(S) || xc1(Sn) || xc2(Sn)
    || xc3(Sn))) {
            state = y11;
            break;
        }
        if (xpy(Z) && xc4(Z)) {
            state = y12;
            break;
        }
        zf();
    break;
}

if (oldstate != state) {
    Game.logger.log("Автомат A1 перешел из состояния y" + oldstate + " в y"
    + state);
    switch (state){
        case y11:
            zi(-1);
        break;
        case y12:
            zi(-1);
        break;

        case y13:
            z03();
        break;

        case y14:
        break;
    }
}
```

```
        Game.logger.log(1,"Автомат A1 завершил обработку события ", e);
    }

// Сброс автомата
    public void resetAutomat () {
        state = y11;
        imageState = 0;
    }
}
```

6. Класс *Hole*

```
package loderunner;

import java.awt.*;
import java.util.TreeMap;

/*
 * Хранит элемент Яма.
 */
public class Hole implements Runnable {

    private int x, y, cellSize;
    private static int lifeTime = 5000;
    private static int deathTime = 1200;
    private int image;
    private int iSize;
    public boolean isDead = false;
    private boolean isClosing = false;

    private Level level;
    private static TreeMap holes = new TreeMap();

    public Hole(int y, int x, Level level){
        this.x = x;
        this.y = y;
        cellSize = Level.getCellSize();
        image = 0;
        iSize = Game.holeImages.length;
        this.level = level;
        holes.put(new Integer( x + 100 * y), this);
        (new Thread(this)).start();
    }

    public void draw(Graphics g){
      g.drawImage(Game.holeImages[image], x*cellSize, y*cellSize, cellSize,
      cellSize, null);
    }

    //Определяет является ли переданная позиция(x,y) - позицией ямы
    public boolean isEqual(int x, int y){
        if((this.x == x) && (this.y ==y)) return true;
        else return false;
    }

//Thread ямы
    public void run (){
```

```
        level.setBoxAT(y,x,Level.HOLE);
        try {
            Thread.currentThread().sleep(lifeTime);
        } catch (Exception e) {
            e.printStackTrace();
        }
        isClosing = true;
        for (int i = 0; i < iSize - 1; i++ ) {
            image++;
            try {
                Thread.currentThread().sleep(deathTime / iSize);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        isDead = true;
        holes.remove(new Integer(x + 100 * y));
        level.setBoxAT(y,x,Level.BRICK1);
    }

//Возвращает яму в позиции p, либо null, если е там нет
    public static Hole getHole(Position p){
        return (Hole) holes.get(new Integer(p.getX() + 100 * p.getY()));
    }

//Возвращает true, если яма начала зарастать
    public boolean isClosing(){
        return isClosing;
    }
}
```

7. Класс *Lode*

```
package loderunner;


import java.awt.*;

/*
 * Хранит состояние структуры Lode.
 */
public class Lode {

    private int x,y;
    private boolean isStatic;

    public Lode(int x,int y){
        this.x = x;
        this.y = y;
        isStatic = true;
    }

    //Прорисовка Золота
    public void draw(Graphics g){
        if (isStatic) {
            g.drawImage(Game.lode, x * Level.getCellSize(), y * Level.getCellSize(),
            Level.getCellSize(), Level.getCellSize(),null);
        }
```

```
    }

    //Указывает на то покоится ли золото или переносится врагом.
    public void setStatic(boolean b){
        isStatic = b;
    }

    //Сравнивает две позиции
    public boolean isEqual(int x,int y){
        if (!isStatic) return false;
        if ((this.x == x) && (this.y == y)) return true;
      else return false;
    }

    //Устанавливает позицию золота
    public void setPosition(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

8. Класс *Logger*

```
package loderunner;

import java.text.*;
import java.util.Date;

/**
 * класс Logger - протоколирование игры
 */
public class Logger {

    private boolean writable;
    private SimpleDateFormat sdf;

    public Logger() {
        sdf = new SimpleDateFormat("HH:mm:ss SSS");
        writable = false;
    }

    public void setWritable(boolean b) {
        writable = b;
    }

    public void log (String str) {
        if (writable)
            System.out.println( sdf.format(new Date()) + " - " + str);
    }

    public void log (String str, int e) {
        switch (e) {
            case Game.e10: log (str + "от таймера"); break;
            case Game.e06: log (str + "нажата клавиша ESC"); break;
            case Game.e00: log (str + "нажата другая клавиша"); break;
            case Game.e05: log (str + "нажата клавиша SPACE"); break;
            default: log (str);
        }
```

```
}

public void log (int i, String str) {
    if (i == 1)
        log ("   " + str );
    else if (i == 2)
        log ("       " + str );
    else log(str);
}

public void log (int i, String str, int e) {
    switch (e) {
        case Game.e00: log (i, str + "нажата другая клавиша"); break;
        case Game.e01: log (i, str + "нажата клавиша вверх"); break;
        case Game.e02: log (i, str + "нажата клавиша вниз"); break;
        case Game.e03: log (i, str + "нажата клавиша влево"); break;
        case Game.e04: log (i, str + "нажата клавиша вправо"); break;
        case Game.e05: log (i, str + "нажата клавиша выкопать яму"); break;
        case Game.e06: log (i, str + "нажата клавиша ESC"); break;
        default: log (i, str);
    }
}
```

Приложение 2. Пример протокола при окончании игры

13:57:28 205 - Автомат A0 запущен с событием от таймера
13:57:28 205 - Автомат A1 запущен с событием нажата клавиша вправо
13:57:28 205 - zs - шаг по горизонтали
13:57:28 205 - Автомат A1 завершил обработку события нажата клавиша вправо
13:57:28 205 - z03 - движение врагов
13:57:28 205 - Автомат A1 запущен с событием нажата клавиша вниз
13:57:28 205 - zu - шаг вверх
13:57:28 205 - Автомат A1 завершил обработку события нажата клавиша вниз
13:57:28 205 - Автомат A1 запущен с событием нажата клавиша вниз
13:57:28 205 - zf - один шаг полета
13:57:28 205 - Автомат A1 завершил обработку события нажата клавиша вниз
13:57:28 205 - Автомат A1 запущен с событием нажата клавиша вправо
13:57:28 205 - Автомат A1 перешел из состояния y11 в y14
13:57:28 205 - Автомат A1 завершил обработку события нажата клавиша вправо
13:57:28 205 - Автомат A1 запущен с событием нажата клавиша вправо
13:57:28 205 - zs - шаг по горизонтали
13:57:28 205 - Автомат A1 завершил обработку события нажата клавиша вправо
13:57:28 205 - Автомат A1 запущен с событием нажата клавиша вправо
13:57:28 205 - zs - шаг по горизонтали
13:57:28 205 - Автомат A1 завершил обработку события нажата клавиша вправо
13:57:28 205 - z04 - обновление состояния золота
13:57:28 205 - z02_1 - вывод игрового пространства
13:57:28 221 - Автомат A0 завершил обработку события от таймера
13:57:28 268 - Автомат A0 запущен с событием от таймера
13:57:28 268 - Автомат A1 запущен с событием нажата клавиша вправо
13:57:28 268 - zs - шаг по горизонтали
13:57:28 268 - Автомат A1 завершил обработку события нажата клавиша вправо
13:57:28 268 - z03 - движение врагов
13:57:28 268 - Автомат A1 запущен с событием нажата клавиша вниз
13:57:28 268 - zu - шаг вверх
13:57:28 268 - Автомат A1 перешел из состояния y13 в y11
13:57:28 268 - Автомат A1 завершил обработку события нажата клавиша вниз
13:57:28 268 - Автомат A1 запущен с событием нажата клавиша вниз
13:57:28 565 - Автомат A0 запущен с событием нажата клавиша ESC
13:57:28 581 - z06 - отключение таймера
13:57:28 596 - Автомат A0 перешел из состояния y02 в y01
13:57:28 596 - z00 - создание объекта Level
13:57:28 596 - z07 - установка первого уровня
13:57:28 596 - z02_2 - вывод заставки
13:57:28 612 - zf - один шаг полета
13:57:28 627 - Автомат A1 завершил обработку события нажата клавиша вниз
13:57:28 627 - Автомат A1 запущен с событием нажата клавиша вправо
13:57:28 627 - zf - один шаг полета
13:57:28 627 - Автомат A1 завершил обработку события нажата клавиша вправо
13:57:28 627 - Автомат A1 запущен с событием нажата клавиша вправо
13:57:28 627 - zs - шаг по горизонтали

13:57:28 627 - Автомат A1 завершил обработку события нажата клавиша вправо
13:57:28 627 - Автомат A1 запущен с событием нажата клавиша вправо
13:57:28 627 - zs - шаг по горизонтали
13:57:28 627 - Автомат A1 завершил обработку события нажата клавиша вправо
13:57:28 627 - z04 - обновление состояния золота
13:57:28 627 - z02_1 - вывод игрового пространства
13:57:28 659 - Автомат A0 перешел из состояния y02 в y01
13:57:28 659 - z00 - создание объекта Level
13:57:28 659 - z07 - установка первого уровня
13:57:28 659 - z02_2 - вывод заставки
13:57:28 674 - Автомат A0 завершил обработку события от таймера
13:57:28 674 - Автомат A0 завершил обработку события нажата клавиша ESC