Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра «Компьютерные технологии»

О.Г. Степанов, А.А. Шалыто

Система эмуляции поведения «умной» мухи

Объектно-ориентированное программирование с явным выделением состояний

Проектная документация

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

Санкт-Петербург
2003

Оглавление

Введение

Для алгоритмизации и программирования задач логического управления А.А.Шалыто была предложена SWITCH-технология [1], которая применительно к событийным и объектно-ориентированным программам была развита им совместно с Н.И.Туккелем [2]. Подробно ознакомиться с этой технологией и с конкретными примерами ее использования можно на сайтах http://is.ifmo.ru и http://www.softcraft.ru. Эта технология удобна для задач управления техническими объектами, такими как, например, лифт [3]. Это связано с тем, что при использовании автоматного подхода, в частности, удается повысить централизацию логики управления в программном коде. Другое достоинство этого подхода состоит в том, что код является изоморфным графу переходов, по которому он строился. Это позволяет для понимания логики работы программ (их поведения) не обращаться к текстам программ, а рассматривать лишь графы переходов.

В настоящей работе эта технология использована для создания простейшей игры — управления «умной» мухой [4].

Программа была реализована в виде *Java*-аплета, что позволяет запустить ее на практически любой системе.

В данной работе совместно применяются объектно-ориентированное и автоматное программирование, что названо в работе [5] “объектно-ориентированное программирование с явным выделением состояний”.

1. Постановка задачи

В данной работе выполнена эмуляция поведения «умной» мухи с использованием SWITCH-технологии. На этом примере показана реализация алгоритмов искусственного интеллекта, обеспечивающих решение этой задачи.

За основу взята программа «умная муха», предложенная в работе [4]. В этой программе, написанной на языке *C* под *DOS*, поведение мухи представлено конечным автоматом, реализованным оператором *switch*. Программа рисует на экране две точки, положение одной из которых определяется игроком, а второй – мухой. Точка, соответствующая игроку, управляется клавишами на клавиатуре. Муха летает по экрану, меняя тактику поведения в зависимости от положения точки игрока.

Муха имеет тактику, а стратегии у нее нет. У нее нет и цели. Поэтому ее движения малопонятны и хаотичны, что, впрочем, соответствует поведению реальных мух.

Код разрабатываемой программы базируется на коде-прототипе. Однако прототип имеет ряд недостатков:

- код, реализующий выходные воздействия, размещен непосредственно в состояниях, что делает его громоздким и резко затрудняет понимание;
- часть логики вынесена за рамки оператора *switch*;
- автомат управляется не событиями, а в цикле с задержкой.

Эти недостатки были устранены в данной работе, кроме того, в задаче улучшен интерфейс.

Программа написана на языке *Java* и выполнена в виде аплета. При этом в ней реализованы следующие функции:

- «введена» муха;
- положение игрока задается курсором мыши;
- текущая тактика мухи иллюстрируется цветом, которым она окрашивается;
- при нахождении на большом расстоянии от курсора муха старается подлететь к нему и окрашивается синим цветом (тактика «Преследование»);
- когда муха находится слишком близко к курсору, она старается отлететь от него и окрашивается красным цветом (тактика «Уклонение»);

- когда муха находится на «среднем» расстоянии от курсора, она движется случайным образом и окрашивается зеленым цветом (тактика «Случайное движение»);
- на «среднем» расстоянии муха может также случайно выбрать одну из заранее заданных траекторий полета. При этом она окрашивается желтым цветом (тактика «Движение по траектории»).

Состояния, которые будут использоваться в автомате управления мухой, совпадают с названиями тактик.

2. Диаграмма классов

Несмотря на кажущуюся простоту системы, ее диаграмма классов весьма нетривиальна (Рис. 1).

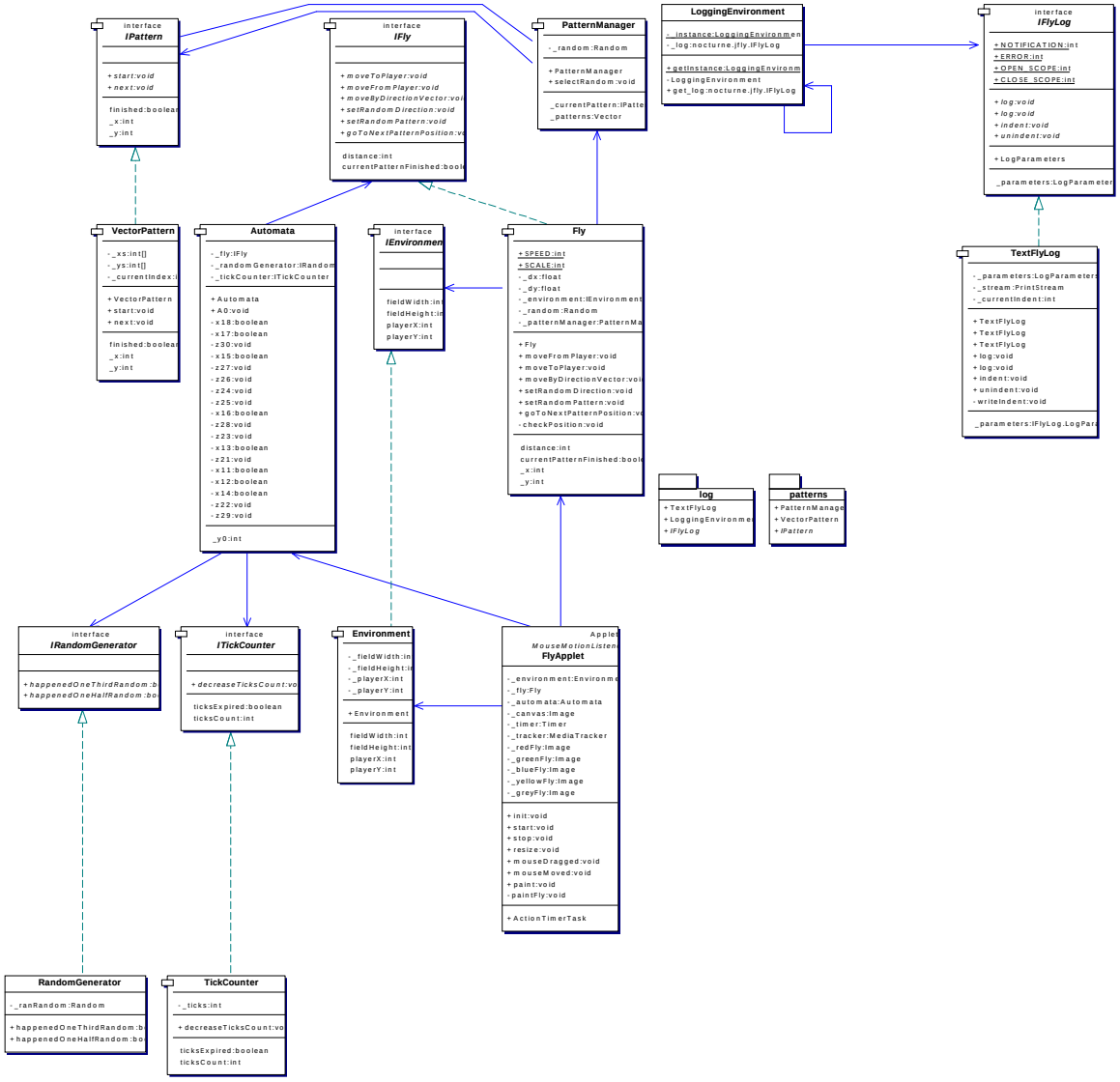

Рис. 1. Диаграмма классов

Диаграмма выполнена в соответствии со стандартом *UML* [6] с помощью CASE-средства *Together Control Center 6*. Бирюзовые линии обозначают обобщение, а синие – связи.

Эта диаграмма содержит десять классов и шесть интерфейсов.

Сначала перечислим классы:

- класс `Automata` реализует автомат, реализующий поведение мухи;
- класс `Fly` реализует «тело» мухи;
- класс `FlyApplet` представляет интерфейс приложения и изображает муху;
- класс `Environment` реализует окружающую среду мухи (напрмер, размеры поля, положение курсора);
- класс `VectorPattern`. Каждый экземпляр этого класса хранит одну векторную траекторию движения мухи;
- класс `PatternManager` хранит траектории движения мухи и позволяет осуществлять движение по каждой из них;
- классы `TextFlyLog` и `LoggingEnvironment` отвечают за протоколирование;
- класс `RandomGenerator` реализует генератор случайных событий;
- класс `TickCounter` реализует счетчик тактов.

Перечислим интерфейсы:

- `IFly` абстрагирует реализацию «тела» мухи;
- `IEnvironment` абстрагирует реализацию среды мухи;
- `IPattern` абстрагирует конкретную реализацию траектории;
- `IFlyLog` абстрагирует реализацию системы протоколирования;
- `IRandomGenerator` абстрагирует генератор случайных событий;
- `ITickCounter` абстрагирует счетчик тактов.

3. Класс «Automata»

3.1. Описание

Класс `Automata` инкапсулирует в себе логику поведения мухи. Он содержит автомат A0, непосредственно реализующий поведение мухи.

3.2. Автомат A0

3.2.1. Описание

Поведение мухи реализовано в автомате A0, входящем в этот класс. Автомат имеет пять состояний:

- «Преследование»;
- «Уклонение»;
- «Случайное движение»;
- «Движение по траектории»;
- «Выбор тактики».

В состоянии «Выбор тактики» автомат, исходя из текущей информации о среде, производит выбор оптимальной тактики. В начале автомат находится в этом состоянии, в котором в зависимости от состояния среды происходит переход в одно из состояний, соответствующих определенной тактике. В этом состоянии муха реализует соответствующую тактику и затем возвращается в исходное состояние.

Обратим внимание, что муха, пребывая в том или ином состоянии, реализует свое поведение по «шагам». Для этого в систему введем таймер, который стробирует выполнение «шагов» движения мухи, сливающиеся в непрерывную траекторию.

Автомат управляется внешним событием *e1* от таймера. При этом если муха находится в состоянии «Выбор тактики», производится ее выбор. Если муха реализует некоторую тактику, то она либо выполняет один шаг, либо возвращается в состояние «Выбор тактики», если реализация тактики завершена.

Счетчик тактов предназначен для обеспечения реализации заданной тактики в течение определенного промежутка времени. Счетчик тактов используется в трех состояниях — «Преследование», «Уклонение» и «Случайное движение». Он используется следующим образом. При переходе из состояния «Выбор тактики» в состояние, непосредственно реализующее тактику, счетчику присваивается начальное значение (от 5 до 20) [4]. Затем каждый такт автомат из одного из указанных выше трех состояний возвращается в то же состояние, реализуя тактику и уменьшая значение счетчика на единицу. Когда значение счетчика сравнивается с нулем, автомат переходит в состояние «Выбор тактики».

В состоянии «Движение по траектории» счетчик тактов не используется. В этом состоянии муха описывает заранее заданную траекторию. После этого автомат возвращается в состояние «Выбор тактики».

К сожалению, выбор многих параметров входных переменных автомата (диапазоны расстояний, вероятности переходов, длительности реализации тактик и т.д.), выполненный в работе [4], сложно объяснить. Это связано с отсутствием стратегической цели поведения мухи. Единственное, что можно отметить — муха «боится» подлетать слишком близко и не отлетает очень далеко.

3.2.2. Диаграмма связей

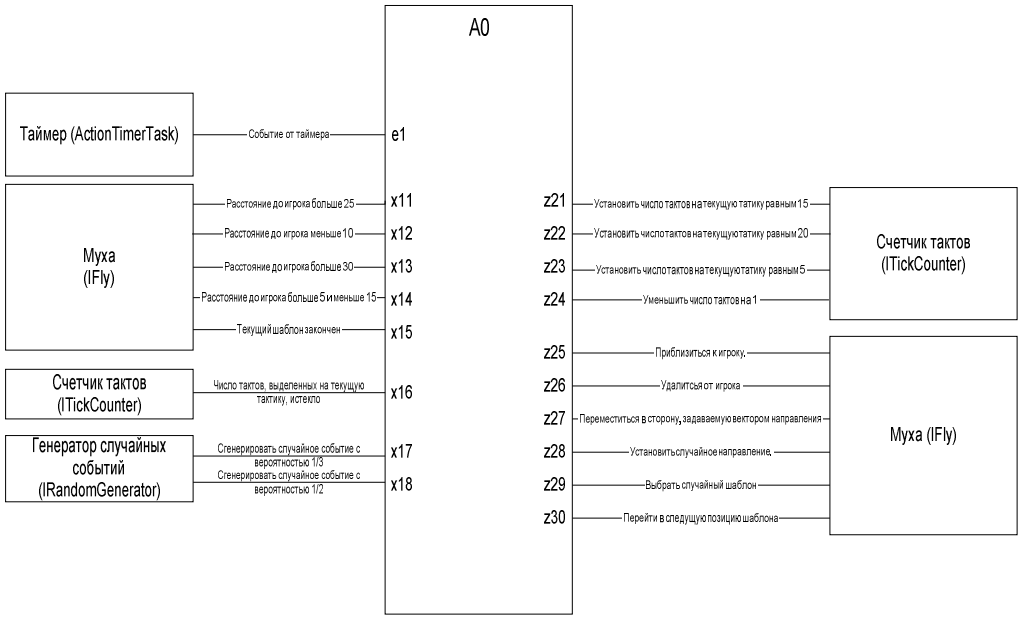

Рис. 2. Диаграмма связей автомата A0

3.2.3. Граф переходов

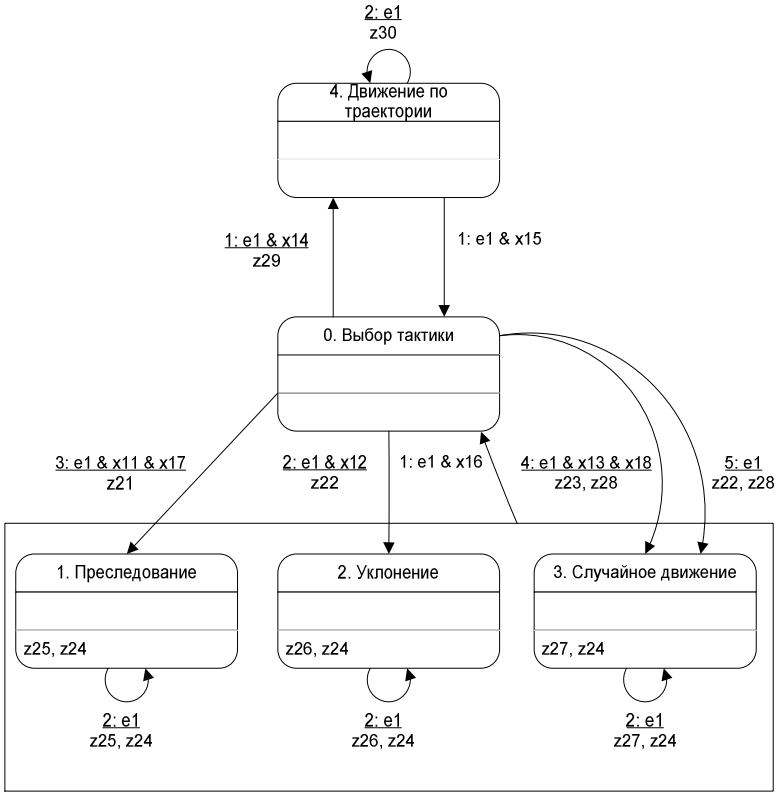

Рис. 3. Граф переходов автомата A0

4. Интерфейс «IFly» и класс «Fly»

Интерфейс `IFly` абстрагирует управляемый объект – муху. Он позволяет проверять все необходимые входные переменные и осуществлять выходные воздействия, связанные с управлением мухой. Класс `Fly` является реализацией этого интерфейса, управляющей виртуальной мухой.

5. Траектории

Важной частью системы является поддержка заранее заданных траекторий. Эта часть логики реализуется классами `PatternManager` и `VectornPattern`. Для абстрагирования от конкретной реализации траектории используется интерфейс `IPattern`.

Экземпляр класса `VectorPattern` представляет собой траекторию, заранее вписанную в код в виде последовательности координат X и Y. Когда мухе подается команда перейти к следующему шагу траектории, она запрашивает координаты этого шага у траектории и перемещается в указанную точку.

Класс `PatternManager` позволяет управлять выбором траектории. Каждый раз, переходя в состояние «Движение по траектории», автомат дает мухе команду выбрать случайную траекторию, которая делегируется классу `PatternManager`.

5. Интерфейс приложения

Интерфейс приложения выполнен в виде аплета и управляет взаимодействием с пользователем. Он обрабатывает события при перемещении курсора мыши, передает событие от таймера автомату A0, управляет рисованием, освобождает занимаемые программой ресурсы компьютера при закрытии главного окна приложения. В аплете также происходит отрисовка мухи в виде окрашенного изображения, цвет которого зависит от текущего состояния:

- серый – «Выбор тактики»;
- красный – «Уклонение»;
- зеленый – «Случайное движение»;
- синий – «Преследование»;
- желтый – «Движение по траектории».

Интерфейс приложения приведен на рис. 4. При этом отметим, что на нем не отображен курсор, относительно которого перемещается муха, так как это не позволяют сделать встроенные средства ОС *Windows*.

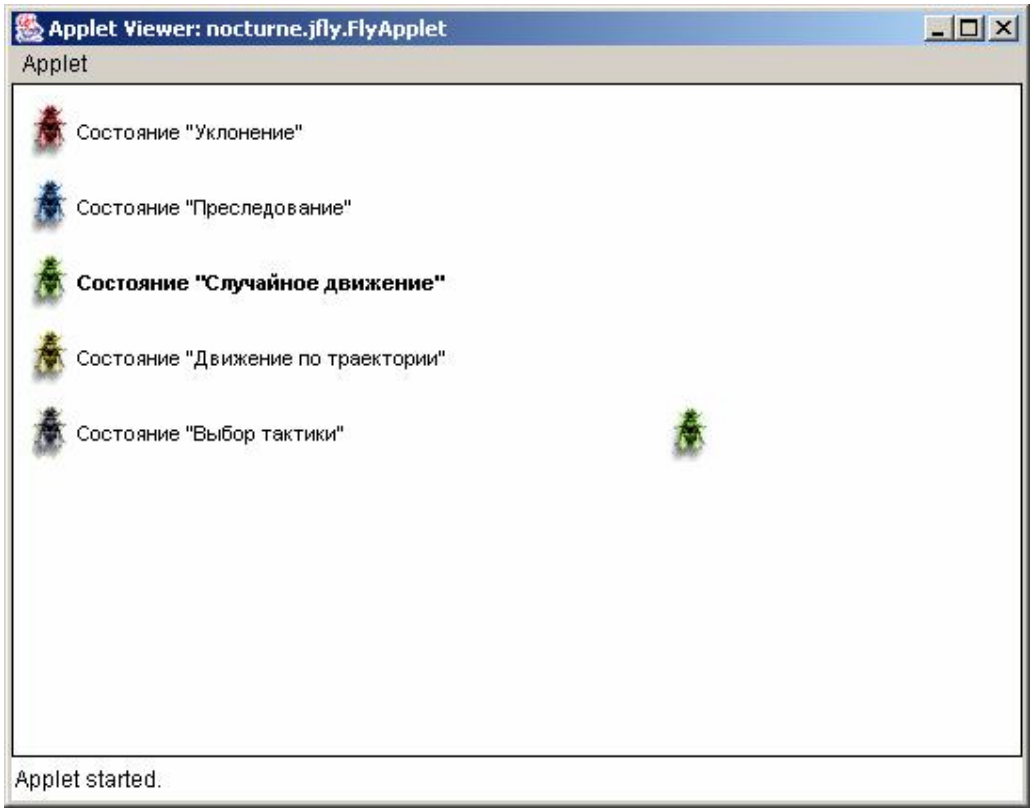

Рис. 4. Интерфейс приложения

6. Протоколирование

Для протоколирования создана специальная библиотека, поддерживающая разные типы сообщений и автоматические отступы. Ниже приведен пример протокола:

```
Fri Feb 21 19:18:47 MSK 2003 { Автомат запущен событием 1 в состоянии 0
Fri Feb 21 19:18:47 MSK 2003   * x14 (Расстояние до игрока больше 5 и меньше 15) =  false
Fri Feb 21 19:18:47 MSK 2003   * x12 (Расстояние до игрока меньше 10) =  false
Fri Feb 21 19:18:47 MSK 2003   * x11 (Расстояние до игрока больше 25) =  true
Fri Feb 21 19:18:47 MSK 2003   * x17 (Герерирует случайное событие с вероятностью 1/3) =  false
Fri Feb 21 19:18:47 MSK 2003   * x17 (Расстояние до игрока больше 30) =  true
Fri Feb 21 19:18:47 MSK 2003   * x18 (Герерирует случайное событие с вероятностью 1/2) =  true
Fri Feb 21 19:18:47 MSK 2003   * z23: Установить число тактов на текущую татику равным 5
Fri Feb 21 19:18:47 MSK 2003   * z28: Устанавливаем случайное направление
Fri Feb 21 19:18:47 MSK 2003   * Автомат перешел в состояние 3
Fri Feb 21 19:18:47 MSK 2003   * z27: Двигаемся в заданную сторону
Fri Feb 21 19:18:47 MSK 2003   * z24: Уменьшить число тактов на 1
Fri Feb 21 19:18:47 MSK 2003 } Обработка события 1 завершена
Fri Feb 21 19:18:47 MSK 2003 { Автомат запущен событием 1 в состоянии 3
Fri Feb 21 19:18:47 MSK 2003   * x16 (Количество тактов, выделенных на данную тактику, истекло) =
false
Fri Feb 21 19:18:47 MSK 2003   * z27: Двигаемся в заданную сторону
Fri Feb 21 19:18:47 MSK 2003   * z24: Уменьшить число тактов на 1
Fri Feb 21 19:18:47 MSK 2003 } Обработка события 1 завершена
Fri Feb 21 19:18:47 MSK 2003 { Автомат запущен событием 1 в состоянии 3
Fri Feb 21 19:18:47 MSK 2003   * x16 (Количество тактов, выделенных на данную тактику, истекло) =
false
Fri Feb 21 19:18:47 MSK 2003   * z27: Двигаемся в заданную сторону
Fri Feb 21 19:18:47 MSK 2003   * z24: Уменьшить число тактов на 1
Fri Feb 21 19:18:47 MSK 2003 } Обработка события 1 завершена
Fri Feb 21 19:18:47 MSK 2003 { Автомат запущен событием 1 в состоянии 3
Fri Feb 21 19:18:47 MSK 2003   * x16 (Количество тактов, выделенных на данную тактику, истекло) =
false
Fri Feb 21 19:18:47 MSK 2003   * z27: Двигаемся в заданную сторону
Fri Feb 21 19:18:47 MSK 2003   * z24: Уменьшить число тактов на 1
Fri Feb 21 19:18:47 MSK 2003 } Обработка события 1 завершена
Fri Feb 21 19:18:47 MSK 2003 { Автомат запущен событием 1 в состоянии 3
Fri Feb 21 19:18:47 MSK 2003   * x16 (Количество тактов, выделенных на данную тактику, истекло) =
false
Fri Feb 21 19:18:47 MSK 2003   * z27: Двигаемся в заданную сторону
Fri Feb 21 19:18:47 MSK 2003   * z24: Уменьшить число тактов на 1
Fri Feb 21 19:18:47 MSK 2003 } Обработка события 1 завершена
Fri Feb 21 19:18:47 MSK 2003 { Автомат запущен событием 1 в состоянии 3
Fri Feb 21 19:18:47 MSK 2003   * x16 (Количество тактов, выделенных на данную тактику, истекло) =
true
Fri Feb 21 19:18:47 MSK 2003   * Автомат перешел в состояние 0
Fri Feb 21 19:18:47 MSK 2003 } Обработка события 1 завершена
Fri Feb 21 19:18:47 MSK 2003 { Автомат запущен событием 1 в состоянии 0
Fri Feb 21 19:18:47 MSK 2003   * x14 (Расстояние до игрока больше 5 и меньше 15) =  false
Fri Feb 21 19:18:47 MSK 2003   * x12 (Расстояние до игрока меньше 10) =  false
Fri Feb 21 19:18:47 MSK 2003   * x11 (Расстояние до игрока больше 25) =  true
Fri Feb 21 19:18:47 MSK 2003   * x17 (Герерирует случайное событие с вероятностью 1/3) =  true
Fri Feb 21 19:18:47 MSK 2003   * z21: Установить число тактов на текущую татику равным 15
Fri Feb 21 19:18:47 MSK 2003   * Автомат перешел в состояние 1
Fri Feb 21 19:18:47 MSK 2003   * z25: Приближаемся к игроку
Fri Feb 21 19:18:47 MSK 2003   * z24: Уменьшить число тактов на 1
Fri Feb 21 19:18:47 MSK 2003 } Обработка события 1 завершена
Fri Feb 21 19:18:47 MSK 2003 { Автомат запущен событием 1 в состоянии 1
Fri Feb 21 19:18:47 MSK 2003   * x16 (Количество тактов, выделенных на данную тактику, истекло) =
false
Fri Feb 21 19:18:47 MSK 2003   * z25: Приближаемся к игроку
Fri Feb 21 19:18:47 MSK 2003   * z24: Уменьшить число тактов на 1
Fri Feb 21 19:18:47 MSK 2003 } Обработка события 1 завершена
Fri Feb 21 19:18:47 MSK 2003 { Автомат запущен событием 1 в состоянии 1
Fri Feb 21 19:18:47 MSK 2003   * x16 (Количество тактов, выделенных на данную тактику, истекло) =
false
Fri Feb 21 19:18:47 MSK 2003   * z25: Приближаемся к игроку
Fri Feb 21 19:18:47 MSK 2003   * z24: Уменьшить число тактов на 1
Fri Feb 21 19:18:47 MSK 2003 } Обработка события 1 завершена
Fri Feb 21 19:18:48 MSK 2003 { Автомат запущен событием 1 в состоянии 1
Fri Feb 21 19:18:48 MSK 2003   * x16 (Количество тактов, выделенных на данную тактику, истекло) =
false
Fri Feb 21 19:18:48 MSK 2003   * z25: Приближаемся к игроку
Fri Feb 21 19:18:48 MSK 2003   * z24: Уменьшить число тактов на 1
Fri Feb 21 19:18:48 MSK 2003 } Обработка события 1 завершена
Fri Feb 21 19:18:48 MSK 2003 { Автомат запущен событием 1 в состоянии 1
```

```
Fri Feb 21 19:18:48 MSK 2003   * x16 (Количество тактов, выделенных на данную тактику, истекло) =
false
Fri Feb 21 19:18:48 MSK 2003   * z25: Приближаемся к игроку
Fri Feb 21 19:18:48 MSK 2003   * z24: Уменьшить число тактов на 1
Fri Feb 21 19:18:48 MSK 2003 } Обработка события 1 завершена
```

Литература

1. *Шалыто А.А.* SWITCH-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998.
2. *Шалыто А.А., Туккель Н.И.* Программирование с явным выделением состояний // Мир ПК, 2001. №8, №9.
3. *Наумов А.С., Шалыто А.А.* Система управления лифтом — http://is.ifmo.ru
4. *Ла Мот А., Ратклифф Д., Семинаторе М.* и др. Секреты программирования игр. СПб.: Питер, 1995.
5. *Туккель Н.И., Шалыто А.А.* Система управления танком для игры Robocode. Объектно-ориентированное программирование с явным выделением состояний. — http://is.ifmo.ru
6. *Ларман К.* Применение UML и шаблонов проектирования. М.:Вильямс, 2001.

Приложение. Исходный код программы

1. Файл «Automata.java»

```
// В этом файле логика автомата A0 выделена жирным шрифтом
package nocturne.jfly;


import java.text.MessageFormat;
import java.util.Vector;
import java.util.Random;

/**
 * Класс “Автомат” A0
 */
public class Automata
{
    // Управлямый объект
    private IFly _fly;

    // Генератор случайных чисел
    private IRandomGenerator _randomGenerator;

    // Счетчик тактов
    private ITickCounter _tickCounter;

    // Конструктор
    public Automata(IFly _fly, IRandomGenerator _randomGenerator, ITickCounter
    _tickCounter)
    {
        this._fly = _fly;
        this._randomGenerator = _randomGenerator;
        this._tickCounter = _tickCounter;
    }

    // Состояние автомата A0
    private int _y0;

    public int get_y0()
    {
        return _y0;
    }

    // Реализация автомата A0
    public void A0(int e)
    {
        LoggingEnvironment.getInstance().get_log().log("A0", MessageFormat.format("Автомат
       запущен событием {0} в состоянии {1}", new Object[] {new Integer(e), new
       Integer(_y0)}), IFlyLog.OPEN_SCOPE);

        // Сохраняется старое состояние
        int oldy0 = _y0;
```

```
switch (_y0)
     {
         case 0:
             if ((e == 1) && x14())
             {
                 z29();_y0 = 4;
             }
             else if ((e == 1) && x12())
             {
                 z22();_y0 = 2;
             }
             else if ((e == 1) && x11() && x17())
             {
                 z21();_y0 = 1;
             }
             else if ((e == 1) && x13() && x18())
             {
                 z23(); z28();_y0 = 3;
             }
             else if (e == 1)
             {
                 z22(); z28();_y0 = 3;
             }
             break;

         case 1:
             if ((e == 1) && x16())
             {
                 _y0 = 0;
             }
             else if (e == 1)
             {
                 z25(); z24();_y0 = 1;
             }
             break;

         case 2:
             if ((e == 1) && x16())
             {
                 _y0 = 0;
             }
             else if (e == 1)
             {
                 z26(); z24();_y0 = 2;
             }
             break;

         case 3:
             if ((e == 1) && x16())
             {
                 _y0 = 0;
             }
             else if (e == 1)
             {
                 z27(); z24(); _y0 = 3;
             }
             break;

         case 4:
             if ((e == 1) && x15())
             {
                 _y0 = 0;
             }
             else if (e == 1)
             {
                 z30();_y0 = 4;
             }
     }

     if (_y0 != oldy0)
     {
         LoggingEnvironment.getInstance().get_log().log("A0",
         MessageFormat.format("Автомат перешел в состояние {0}", new Object[] {new
         Integer(_y0)}), IFlyLog.NOTIFICATION);
```

```
            switch (_y0)
            {
                case 1:
                    z25();z24();
                    break;

                case 2:
                    z26();z24();
                    break;

                case 3:
                    z27();z24();
                    break;
            }
        }

        LoggingEnvironment.getInstance().get_log().log("A0",
        MessageFormat.format("Обработка события {0} завершена", new Object[] {new
        Integer(e)}), IFlyLog.CLOSE_SCOPE);
    }

// Входные переменные

    /**
     * Расстояние до игрока больше 25
     */
    private boolean x11()
    {
        boolean result = _fly.getDistance() > 25;

        LoggingEnvironment.getInstance().get_log().log("A0", MessageFormat.format("x11
        (Расстояние до игрока больше 25) =  {0}", new Object[] {new Boolean(result)}),
        IFlyLog.NOTIFICATION);

        return result;
    }

    /**
     * Расстояние до игрока меньше 10
     */
    private boolean x12()
    {
        boolean result = _fly.getDistance() < 10;

        LoggingEnvironment.getInstance().get_log().log("A0", MessageFormat.format("x12
        (Расстояние до игрока меньше 10) =  {0}", new Object[] {new Boolean(result)}),
        IFlyLog.NOTIFICATION);

        return result;
    }

    /**
     * Расстояние до игрока больше 30
     */
    private boolean x13()
    {
        boolean result = _fly.getDistance() > 30;

        LoggingEnvironment.getInstance().get_log().log("Fly",
        MessageFormat.format("happenedOneThirdRandom (Расстояние до игрока больше 30) =
        {0}", new Object[] {new Boolean(result)}), IFlyLog.NOTIFICATION);

        return result;
    }

    /**
     * Расстояние до игрока больше 5 и меньше 15
     */
    private boolean x14()
    {
        boolean result = _fly.getDistance() > 5 && _fly.getDistance() < 15;

        LoggingEnvironment.getInstance().get_log().log("A0", MessageFormat.format("x14
        (Расстояние до игрока больше 5 и меньше 15) =  {0}", new Object[] {new
        Boolean(result)}), IFlyLog.NOTIFICATION);

        return result;
    }
```

```
    /**
     * Текущий траектория завершена
     */
    private boolean x15()
    {
        return _fly.isCurrentPatternFinished();
    }

    /**
     * Количество тактов, выделенных на данную тактику, истекло
     */
    private boolean x16()
    {
        return _tickCounter.isTicksExpired();
    }

    /**
     * Геререрует случайное событие с вероятностью 1/3
     */
    private boolean x17()
    {
        return _randomGenerator.happenedOneThirdRandom();
    }

    /**
     * Геререрует случайное событие с вероятностью 1/2
     */
    private boolean x18()
    {
        return _randomGenerator.happenedOneHalfRandom();
    }

// Выходные воздействия

    /**
     * Установить число тактов на текущую тактику равным 15
     */
    private void z21()
    {
        LoggingEnvironment.getInstance().get_log().log("A0", "z21: Установить число тактов
       на текущую татику равным 15", IFlyLog.NOTIFICATION);

        _tickCounter.setTicksCount(15);
    }

    /**
     * Установить число тактов на текущую тактику равным 20
     */
    private void z22()
    {
        LoggingEnvironment.getInstance().get_log().log("A0", "z22: Установить число тактов
         на текущую татику равным 20", IFlyLog.NOTIFICATION);

        _tickCounter.setTicksCount(20);
    }

    /**
     * Установить число тактов на текущую тактику равным 5
     */
    private void z23()
    {
        LoggingEnvironment.getInstance().get_log().log("A0", "z23: Установить число тактов
       на текущую татику равным 5", IFlyLog.NOTIFICATION);

        _tickCounter.setTicksCount(5);
    }

    /**
     * Уменьшить число тактов на 1
     */
    private void z24()
    {
        _tickCounter.decreaseTicksCount();
    }
```

```
    /**
     * Приблизиться к игроку
     */
    private void z25()
    {
        _fly.moveToPlayer();
    }

    /**
     * Удалитсья от игрока
     */
    private void z26()
    {
        _fly.moveFromPlayer();
    }

    /**
     * Переместиться в сторону, задаваемую вектором направления
     */
    private void z27()
    {
        _fly.moveByDirectionVector();
    }

    /**
     * Устновить случайное направление
     */
    private void z28()
    {
        _fly.setRandomDirection();
    }

    /**
     * Выбрать случайную траекторию
     */
    private void z29()
    {
        _fly.setRandomPattern();
    }

    /**
     * Перейти в следущую позицию шаблона
     */
    private void z30()
    {
        _fly.goToNextPatternPosition();
    }

}
```

2. Файл «Environment.java»

```
package nocturne.jfly;

/**
 * Среда обитания мухи
 */
public class Environment implements IEnvironment
{
    // Размеры поля
    private int _fieldWidth;
    private int _fieldHeight;

    // Координаты игрока
    private int _playerX;
    private int _playerY;

    public Environment(int _fieldWidth, int _fieldHeight, int _playerX, int _playerY)
    {
        this._fieldWidth = _fieldWidth;
        this._fieldHeight = _fieldHeight;
        this._playerX = _playerX;
        this._playerY = _playerY;
    }
```

```
    /**
     * Возвращает ширину поля
     */
    public int getFieldWidth()
    {
        return _fieldWidth;
    }

    /**
     * Возвращает высоту поля
     */
    public int getFieldHeight()
    {
        return _fieldHeight;
    }

    /**
     * Возвращает X-координату игрока
     */
    public int getPlayerX()
    {
        return _playerX;
    }

    /**
     * Возвращает Y-координату игрока
     */
    public int getPlayerY()
    {
        return _playerY;
    }

    /**
     * Устанавливает X-координату игрока
     */
    public void setPlayerX(int playerX)
    {
        this._playerX = playerX;
    }

    /**
     * Устанавливает Y-координату игрока
     */
    public void setPlayerY(int playerY)
    {
        this._playerY = playerY;
    }
}
```

3. Файл «Fly.java»

```
package nocturne.jfly;


import java.util.Random;
import java.text.MessageFormat;

/**
 * Реализация класса Муха
 */
public class Fly implements IFly
{
    public static int SPEED = 2;
    public static int SCALE = 3;

    // Координаты мухи
    private int _x = 0;
    private int _y = 0;

    // Вектор направления движения мухи
    private float _dx = 0;
    private float _dy = 0;

    // Среда обитания мухи
    private IEnvironment _environment;
```

```
    // Герератор случайных чисел
    private Random _random = new Random();

    // Менеджер траекторий
    private PatternManager _patternManager = new PatternManager();

    public Fly(IEnvironment _environment)
    {
        this._environment = _environment;

        _x = _environment.getFieldWidth() / 2;
        _y = _environment.getFieldHeight() / 2;

        _patternManager.get_patterns().add(
new VectorPattern(
new int[] {1, 1, 1, 1, 1, 2, 2, -1, -2, -3, -1, 0, 0, 1, 2, 2, -2, -2, -1, 0},
new int[] {0, 0, 0, 0, -1, -1, -1, -1, -1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2}));
        _patternManager.get_patterns().add(
new VectorPattern(
new int[] {0, 0, 1, 2, 3, 4, 5, 4, 3, 2, 1, 3, 3, 3, 3, 2, 1, -2, -2, -1},
new int[] {1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 0, 0, 0, 0}));
        _patternManager.get_patterns().add(
new VectorPattern(
new int[] {0, -1, -2, -3, -3, -2, -2, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1},
new int[] {1, 1, 1, 2, 2, -1, -1, -1, -2, -2, -1, -1, 0, 0, 0, 1, 1, 1, 1, 1}));
    }

    // Возвращает расстояние от игрока до мухи
    public int getDistance()
    {
        return    (int)(Math.sqrt((_x    -    _environment.getPlayerX())    *    (_x    -
       _environment.getPlayerX()) +
                          (_y     -     _environment.getPlayerY())     *     (_y     -
       _environment.getPlayerY())) / SCALE);
    }

    /**
     * Текущая траектория завершена
     */
    public boolean isCurrentPatternFinished()
    {
        return _patternManager.get_currentPattern().isFinished();
    }

    /**
     * Удалиться от игрока
     */
    public void moveFromPlayer()
    {
        LoggingEnvironment.getInstance().get_log().log("Fly", "moveFromPlayer: Удаляемся
       от игрока", IFlyLog.NOTIFICATION);


        if (_x < _environment.getPlayerX())
            _x -= SPEED;
        else if (_x > _environment.getPlayerX())
            _x += SPEED;

        if (_y < _environment.getPlayerY())
            _y -= SPEED;
        else if (_y > _environment.getPlayerY())
            _y += SPEED;

        checkPosition();
    }

    /**
     * Приблизиться к игроку
     */
    public void moveToPlayer()
    {
        LoggingEnvironment.getInstance().get_log().log("Fly", "moveToPlayer: Приближаемся
       к игроку", IFlyLog.NOTIFICATION);

        if (_x < _environment.getPlayerX())
            _x += SPEED;
        else if (_x > _environment.getPlayerX())
            _x -= SPEED;

        if (_y < _environment.getPlayerY())
```

```
            _y += SPEED;
        else if (_y > _environment.getPlayerY())
            _y -= SPEED;

        checkPosition();
    }

    /**
     * Переместиться в сторону, задаваемую вектором направления
     */
    public void moveByDirectionVector()
    {
        LoggingEnvironment.getInstance().get_log().log("Fly", "moveByDirectionVector:
        Двигаемся в заданную сторону", IFlyLog.NOTIFICATION);

        _x += _dx;
        _y += _dy;

        checkPosition();
    }

    /**
     * Уствновить случайное направление
     */
    public void setRandomDirection()
    {
        LoggingEnvironment.getInstance().get_log().log("Fly", "setRandomDirection:
        Устанавливаем случайное направление", IFlyLog.NOTIFICATION);

        _dx = _random.nextFloat() * SPEED;
        _dy = _random.nextFloat() * SPEED;
    }

    /**
     * Выбрать случайную траекторию
     */
    public void setRandomPattern()
    {
        _patternManager.selectRandom();
    }

    /**
     * Перейти в следующую позицию траектории
     */
    public void goToNextPatternPosition()
    {
        _x += _patternManager.get_currentPattern().get_x() * SPEED;
        _y += _patternManager.get_currentPattern().get_y() * SPEED;

        _patternManager.get_currentPattern().next();
    }

    public int get_x()
    {
        return _x;
    }

    public int get_y()
    {
        return _y;
    }

    // Находится ли муха на поле
    private void checkPosition()
    {
        if (_x < 0)
            _x = 0;
        if (_y < 0)
            _y = 0;

        if (_x >= _environment.getFieldWidth())
            _x = _environment.getFieldWidth() - 1;

        if (_y >= _environment.getFieldHeight())
            _y = _environment.getFieldHeight() - 1;
    }
}
```

4. Файл «FlyApplet.java»

```
package nocturne.jfly;

import java.applet.Applet;
import java.awt.event.MouseMotionListener;
import java.awt.event.MouseEvent;
import java.awt.*;
import java.util.Timer;
import java.util.TimerTask;

/**
 * Основной аплет
 */
public class FlyApplet extends Applet implements MouseMotionListener
{
    private Environment _environment;
    private Fly _fly;
    private Automata _automata;
    private Image _canvas;
    private Timer _timer = new Timer();
    private MediaTracker _tracker;

    /*
     * Изображения мухи
     */
    private Image _redFly, _greenFly, _blueFly, _yellowFly, _greyFly;

    public void init()
    {
        super.init();

        _environment = new Environment(getWidth(), getHeight(), 0, 0);
        _fly = new Fly(_environment);
        _automata = new Automata(_fly, new RandomGenerator(), new TickCounter());

        _canvas = this.createImage(getWidth(), getHeight());

        _tracker = new MediaTracker(this);
        _redFly = getImage(getDocumentBase(), "img/fly_red.gif");
        _blueFly = getImage(getDocumentBase(), "img/fly_blue.gif");
        _greenFly = getImage(getDocumentBase(), "img/fly_green.gif");
        _yellowFly = getImage(getDocumentBase(), "img/fly_yellow.gif");
        _greyFly = getImage(getDocumentBase(), "img/fly.gif");

        _tracker.addImage(_redFly, 0);
        _tracker.addImage(_blueFly, 1);
        _tracker.addImage(_greenFly, 2);
        _tracker.addImage(_yellowFly, 3);
        _tracker.addImage(_greyFly, 4);

        this.addMouseMotionListener(this);
    }

    public void start()
    {
        super.start();

        try
        {
            _tracker.waitForAll();
        }
        catch (InterruptedException e)
        {
            e.printStackTrace();
        }

        _timer.scheduleAtFixedRate(new ActionTimerTask(), 100, 50);
    }

    public void stop()
    {
        _timer.cancel();

        super.stop();
    }

    public void resize(Dimension d)
    {
        super.resize(d);
```

```
    }

    public void mouseDragged(MouseEvent e)
    {
    }

    public void mouseMoved(MouseEvent e)
    {
        _environment.setPlayerX((int)e.getPoint().getX());
        _environment.setPlayerY((int)e.getPoint().getY());
    }

    public void paint(Graphics g)
    {
        update(g);
    }

    public void update(Graphics g)
    {
        g.drawImage(_canvas, 0, 0, this);
    }

    private void paintFly()
    {
        Graphics g = _canvas.getGraphics();

        g.setColor(Color.WHITE);
        g.fillRect(0, 0, _canvas.getWidth(this), _canvas.getHeight(this));
        g.setColor(Color.BLACK);
        g.drawRect(0, 0, getWidth() - 1, getHeight() - 1);

        // Изображение мухи
        Image flyImage = null;

        switch (_automata.get_y0())
        {
        case 0:
            flyImage = _greyFly;
            break;

        case 1:
            flyImage = _blueFly;
            break;

        case 2:
            flyImage = _redFly;
            break;

        case 3:
            flyImage = _greenFly;
            break;

        case 4:
            flyImage = _yellowFly;
            break;
        }

        int x = _fly.get_x() - flyImage.getWidth(this) / 2;
        int y = _fly.get_y() - flyImage.getHeight(this) / 2;

        g.drawImage(flyImage, x, y, this);

         // Поясняющие надписи
         int imagey = 10;

         imagey = drawFly(g, flyImage, _redFly, "Состояние \"Уклонение\"", imagey);
         imagey = drawFly(g, flyImage, _blueFly, "Состояние \"Преследование\"", imagey);
         imagey = drawFly(g, flyImage, _greenFly, "Состояние \"Случайное движение\"",
         imagey);
         imagey = drawFly(g, flyImage, _yellowFly, "Состояние \"Движение по траектории\"",
         imagey);
         imagey = drawFly(g, flyImage, _greyFly, "Состояние \"Выбор тактики\"", imagey);
    }

    private int drawFly( Graphics g, Image flyImage, Image sampleImage, String text, int y
)
    {
        g.drawImage(sampleImage, 10, y, this);
        y += sampleImage.getHeight(this) + 10;
```

```
            if (flyImage.equals(sampleImage))
                g.setFont(_boldFont);
            else
                g.setFont(_normalFont);

            g.drawString(text, 10 + sampleImage.getWidth(this) + 5, y - 20);

            return y;
        }

        /**
         * Класс, обновляющий поле
         */
        public class ActionTimerTask extends TimerTask
        {
            /**
             * Действия на каждом шаге аплета
             */
            public void run()
            {
                _automata.A0(1);

                paintFly();
                update(getGraphics());
            }
        }
    }
```

5. Файл «IEnvironment.java»

```
package nocturne.jfly;

/**
 * Определяет среду мухи
 */
public interface IEnvironment
{
    /**
     * Возвращает ширину поля
     */
    public int getFieldWidth();

    /**
     * Возвращает высоту поля
     */
    public int getFieldHeight();

    /**
     * Возвращает X-координату игрока
     */
    public int getPlayerX();

    /**
     * Возвращает Y-координату игрока
     */
    public int getPlayerY();
}
```

6. Файл «IFly.java»

```
package nocturne.jfly;

/**
 * Интерфейс IFly абстрагирует управляемый объект
 */
public interface IFly
{
    /**
     *  Возвращает расстояние от игрока до мухи
     */
    public int getDistance();

    /**
     * Текущая траектория завершена
     */
    public boolean isCurrentPatternFinished();
```

```
    /**
     * Приблизиться к игроку
     */
    public void moveToPlayer();

    /**
     * Удалитсья от игрока
     */
    public void moveFromPlayer();

    /**
     * Переместиться в сторону, задаваемую вектором направления
     */
    public void moveByDirectionVector();

    /**
     * Уствновить случайное направление
     */
    public void setRandomDirection();

    /**
     * Выбрать случайную траекторию
     */
    public void setRandomPattern();

    /**
     * Перейти в следующую позицию траектории
     */
    public void goToNextPatternPosition();
}
```

7. Файл «IRandomGenerator.java»

```
package nocturne.jfly;

/**
 * Представляет абстрактный генератор случайных событий
 */
public interface IRandomGenerator
{
    /**
     * Генерирует случайное событие с вероятностью 1/3
     */
    public boolean happenedOneThirdRandom();

    /**
     * Генерирует случайное событие с вероятностью 1/2
     */
    public boolean happenedOneHalfRandom();
}
```

8. Файл «ITickCounter.java»

```
package nocturne.jfly;

/**
 * Представляет абстрактный счетчик тактов
 */
public interface ITickCounter
{
    /**
     * Количество тактов, выделенных на данную тактику, истекло
     */
    public boolean isTicksExpired();

    /**
     * Устанавливает счетчик тактов
     */
    public void setTicksCount( int ticks );

    /**
     * Уменьшить число тактов на 1
     */
```

```
    public void decreaseTicksCount();
}
```

9. Файл «RandomGenerator.java»

```
package nocturne.jfly;


import java.util.Random;
import java.text.MessageFormat;

/**
 * Генератор случайных событий
 */
public class RandomGenerator implements IRandomGenerator
{
    private Random _ranRandom = new Random();
```

```
    /**
     * Генерирует случайное событие с вероятностью 1/3
     */
    public boolean happenedOneThirdRandom()
    {
        boolean result = _ranRandom.nextDouble() >= 1.0 / 3;

        LoggingEnvironment.getInstance().get_log().log("Fly",
        MessageFormat.format("happenedOneThirdRandom (Геререрирует случайное событие с
        вероятностью 1/3) =  {0}", new Object[] {new Boolean(result)}),
        IFlyLog.NOTIFICATION);

        return result;
    }

    /**
     * Генерирует случайное событие с вероятностью 1/2
     */
    public boolean happenedOneHalfRandom()
    {
        boolean result = _ranRandom.nextDouble() >= 1.0 / 2;

        LoggingEnvironment.getInstance().get_log().log("Fly",
        MessageFormat.format("happenedOneHalfRandom (Геререрирует случайное событие с
        вероятностью 1/2) =  {0}", new Object[] {new Boolean(result)}),
        IFlyLog.NOTIFICATION);

        return result;
    }
}
```

10. Файл «TickCounter.java»

```
package nocturne.jfly;


import java.text.MessageFormat;

/**
 * Счетчик тактов
 */
public class TickCounter implements ITickCounter
{
    private int _ticks = 0;

    /**
     * Количество тактов, выделенных на данную тактику, истекло
     */
    public boolean isTicksExpired()
    {
        boolean result = _ticks == 0;

        LoggingEnvironment.getInstance().get_log().log("Fly",
        MessageFormat.format("isTicksExpired (Количество тактов, выделенных на данную
        тактику, истекло) =  {0}", new Object[] {new Boolean(result)}),
        IFlyLog.NOTIFICATION);

        return result;
    }

    /**
     * Устанавливает счетчик тактов
     */
    public void setTicksCount(int ticks)
    {
        _ticks = ticks;
    }
```

```
    /**
     * Уменьшить число тактов на 1
     */
    public void decreaseTicksCount()
    {
        LoggingEnvironment.getInstance().get_log().log("TickCounter", "decreaseTicksCount:
       Уменьшить число тактов на 1", IFlyLog.NOTIFICATION);

        _ticks--;
    }
}
```

11. Файл «IPattern.java»

```
package nocturne.jfly;

/**
 * Траектория движения мухи
 */
public interface IPattern
{
    /**
     * Начало движения по траектории
     */
    public void start();

    /**
     * Переход к следующему шагу
     */
    public void next();

    /**
     * Траектория завершена
     */
    public boolean isFinished();

    /**
     * Получить X-координату шага траектории
     */
    public int get_x();

    /**
     * Получить Y-координату шага траектории
     */
    public int get_y();
}
```

12. Файл «PatternManager.java»

```
package nocturne.jfly;

import java.util.Vector;
import java.util.Random;

/**
 * Управляет траекториями поведения
 */
public class PatternManager
{
    // Список траекторий
    private Vector _patterns = new Vector();

    // Текущая траектория
    private IPattern _currentPattern;

    // Генератор случайных чисел
    private Random _random = new Random();
    private IPattern lnkIPattern;

    public IPattern get_currentPattern()
    {
        return _currentPattern;
    }
```

```
    public Vector get_patterns()
    {
        return _patterns;
    }

    public PatternManager()
    {
    }

    /**
     * Выбрать случайную траекторию
     */
    public void selectRandom()
    {
        _currentPattern = (IPattern) _patterns.get(_random.nextInt(_patterns.size()));
        _currentPattern.start();
    }
}
```

13. Файл «VectorPattern.java»

```
package nocturne.jfly;

import java.util.Vector;

/**
 * Векторная траектория
 */
public class VectorPattern implements IPattern
{
    // Векторы координат
    private int[] _xs;
    private int[] _ys;

    // Текущая позиция траектории
    private int _currentIndex = 0;

    public VectorPattern(int[] xs, int[] ys) throws IllegalArgumentException
    {
        if (xs.length != ys.length)
            throw new IllegalArgumentException("Массивы должны быть одного размера");

        this._xs = xs;
        this._ys = ys;
    }

    /**
     * Начало движения по траектории
     */
    public void start()
    {
        _currentIndex = 0;
    }

    /**
     * Переход к следующему шагу
     */
    public void next()
    {
        if (_currentIndex < _xs.length - 1)
            _currentIndex++;
    }

    /**
     * Траектория завершена
     */
    public boolean isFinished()
    {
        return _currentIndex == _xs.length - 1;
    }
```

```
    /**
     * Получить X-координату шага траектории
     */
    public int get_x()
    {
        return _xs[_currentIndex];
    }

    /**
     * Получить Y-координату шага траектории
     */
    public int get_y()
    {
        return _ys[_currentIndex];
    }
}
```

14. Файл «IFlyLog.java»

```
package nocturne.jfly;

public interface IFlyLog
{
    /**
     * Задает параметры протоколирования
     */
    public class LogParameters
    {
        /**
         * Отступ
         */
        private int _indent = 2;

        /**
         * Указывать ли маркер времени
         */
        private boolean _writeTimestamp = true;

        /**
         * Автоизменение отступа при входе/выходе из области
         */
        private boolean _scopeAutoIndent = true;

        public int get_indent()
        {
            return _indent;
        }

        public void set_indent(int _indent)
        {
            this._indent = _indent;
        }

        public boolean is_writeTimestamp()
        {
            return _writeTimestamp;
        }

        public void set_writeTimestamp(boolean _writeTimestamp)
        {
            this._writeTimestamp = _writeTimestamp;
        }

        public boolean is_scopeAutoIndent()
        {
            return _scopeAutoIndent;
        }

        public void set_scopeAutoIndent(boolean _scopeAutoIndent)
        {
            this._scopeAutoIndent = _scopeAutoIndent;
        }
```

```
        public LogParameters(int _indent, boolean _writeTimestamp, boolean
         _scopeAutoIndent)
        {
            this._indent = _indent;
            this._writeTimestamp = _writeTimestamp;
            this._scopeAutoIndent = _scopeAutoIndent;
        }

        public LogParameters()
        {
        }
    }

    /*
     * Типы записей
     */

    // Информационная
    public final static int NOTIFICATION = 1;

    // Ошибка
    public final static int ERROR = 2;

    // Начало области
    public final static int OPEN_SCOPE = 3;

    // Конец области
    public final static int CLOSE_SCOPE = 4;

    /**
     * Устанавливает параметры протоколирования
     */
    public void set_parameters( LogParameters parameters );

    /**
     * Получает параметры протоколирования
     */
    public LogParameters get_parameters();

    /**
     * Добавляет запись в протокол
     */
    public void log( String automata, String text );

    /**
     * Добавляет запись в протокол
     */
    public void log( String automata, String text, int type );

    /**
     * Увеличивает отступ
     */
    public void indent();

    /**
     * Уменьшает отступ
     */
    public void unindent();
}
```

15. Файл «LoggingEnvironment.java»

```
package nocturne.jfly;

/**
 * Среда для поддержки протоколирования
 */
public class LoggingEnvironment
{
    private static LoggingEnvironment _instance;

    public static LoggingEnvironment getInstance()
    {
        if (_instance == null)
            _instance = new LoggingEnvironment();

        return _instance;
```

```
    }

    private nocturne.jfly.IFlyLog _log;

    private LoggingEnvironment()
    {
        _log = new TextFlyLog();
    }

    public nocturne.jfly.IFlyLog get_log()
    {
        return _log;
    }
}
```

16. Файл «TextFlyLog.java»

```
package nocturne.jfly;

import java.io.PrintStream;
import java.text.Format;
import java.text.MessageFormat;
import nocturne.jfly.IFlyLog;

/**
 * Класс для ведения текстового протокола
 */
public class TextFlyLog implements nocturne.jfly.IFlyLog
{
    // Параметры протоколирования
    private LogParameters _parameters;

    // Поток для вывода
    private PrintStream _stream;

    // Текущий отступ
    private int _currentIndent;

    public TextFlyLog(IFlyLog.LogParameters _parameters, PrintStream _stream)
    {
        this._parameters = _parameters;
        this._stream = _stream;
    }

    public TextFlyLog(PrintStream _stream)
    {
        this._stream = _stream;
        this._parameters = new LogParameters();
    }

    public TextFlyLog()
    {
        this._stream = System.out;
        this._parameters = new LogParameters();
    }

    /**
     * Устанавливает параметры протоколирования
     */
    public void set_parameters(IFlyLog.LogParameters parameters)
    {
        _parameters = parameters;
    }

    /**
     * Получает параметры протоколирования
     */
    public IFlyLog.LogParameters get_parameters()
    {
        return _parameters;
    }

    /**
     * Добавляет запись в протокол
     */
    public void log(String automata, String text)
    {
```

```
        log(automata, text, NOTIFICATION);
    }

    /**
     * Добавляет запись в протокол
     */
    public void log(String automata, String text, int type)
    {
        String typeSymbol = "";

        if (_parameters.is_writeTimestamp())
        {
            _stream.print(MessageFormat.format("{0} ", new Object[] {new
            java.util.Date().toString()}));
        }

        switch (type)
        {
        case NOTIFICATION:
                typeSymbol = "*";

                writeIndent();
                _stream.print(typeSymbol);

                break;
        case ERROR:
                typeSymbol = "!";

                writeIndent();
                _stream.print(typeSymbol);


                break;
        case OPEN_SCOPE:
                typeSymbol = "{";

                writeIndent();
                _stream.print(typeSymbol);

                if (_parameters.is_scopeAutoIndent())
                    indent();

                break;
        case CLOSE_SCOPE:
                typeSymbol = "}";

                if (_parameters.is_scopeAutoIndent())
                    unindent();

                writeIndent();
                _stream.print(typeSymbol);

                break;
        }

        _stream.println(MessageFormat.format(" {0}", new Object[] {text}));
    }

    /**
     * Увеличивает отступ
     */
    public void indent()
    {
        _currentIndent += _parameters.get_indent();
    }

    /**
     * Уменьшает отступ
     */
    public void unindent()
    {
        _currentIndent -= _parameters.get_indent();
    }

    private void writeIndent()
    {
        for (int i = 0; i < _currentIndent; i++)
            _stream.print(" ");
    }
}
```