Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра “Компьютерные технологии”

Н. Н. Красильников, А. А. Шалыто

Мультиагентная система дорожного движения. Реализация на языке *Java* и текстовом языке автоматного программирования

Проектная документация

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

Санкт-Петербург
2007

Оглавление

Введение

Данная работа основывается на работе “Мультиагентная система дорожного движения” [1].

Целью настоящей работы является изучение **текстового языка автоматного программирования** и возможностей его применения на примере системы дорожного движения.

1. Постановка задачи

Цель работы – создание мультиагентной системы дорожного движения автомобилей на языке *Java* и текстовом языке автоматного программирования.

Также как и в работе “Мультиагентная система дорожного движения” автомобили будут ездить по улицам города с соблюдением правил дорожного движения.

В данной работе взята упрощенная схема такого движения: дороги имеют по одной полосе движения в каждую сторону, которые разделены двойной полосой.

2. Текстовый язык автоматного программирования

Текстовый язык автоматного программирования, описываемый в следующем разделе, предназначен для формальной записи автоматов в виде близком к представлению автоматов в *SWITCH*-технологии [2, 3].

При использовании *SWITCH*-технологии реализация автоматов должна быть изоморфна графу переходов, однако на практике по коду программы тяжело понять действительно ли это так. Компилятор не способен понять, что во время реализации была допущена ошибка. Например, можно пропустить всего один оператор `break` и программа станет некорректной.

Реализация автомата на рассматриваемом языке автоматного программирования позволяет гарантировать изоморфность графу переходов, избавляя, тем самым, программиста от целого класса ошибок. Хотя, можно было и раньше генерировать код автомата по графу переходов (например, с помощью инструментального средства *UniMod* [4]), значительно уменьшая вероятность подобных ошибок. Однако графическое описание автоматов не является единственным, и их текстовое представление может быть весьма полезным для практического использования.

Поскольку рассматриваемый язык автоматного программирования не обладает вычислительной мощностью машины Тьюринга, то писать программы только на нем нельзя. Для того, чтобы написать программу в целом, необходима интеграция с каким-либо универсальным языком программирования. В данном случае это язык *Java*.

Таким образом, при использовании рассматриваемого языка удается сократить семантический пробел между графом переходов автомата и его реализацией на традиционном императивном языке программирования.

3. Реализация текстового языка автоматного программирования

Рассматриваемый текстовый язык автоматного программирования реализован в среде *MPS* (*Meta Programming System*) [5], разработанной в компании *JetBrains*.

Традиционно программа пишется в любом текстовом редакторе, а затем компилируется. Одним из этапов компиляции является синтаксический анализ, в ходе которого строится абстрактное синтаксическое дерево.

В *MPS* фаза синтаксического анализа отсутствует [6, 7], так как пользователь с помощью специального редактора может строить только синтаксически правильные конструкции. Таким образом, на самом деле пользователь сам строит абстрактное синтаксическое дерево.

3.1. Краткое описание синтаксиса

Опишем синтаксис текстового языка автоматного программирования.

Основные конструкции языка:

- **event <type> <name> () {<statements>}**

С помощью этой конструкции описываются возможные события.

Например:

```
event void e0 ( ) {<no statements>}
```

- **initial state <name> {<statements>}**

Начальное состояние автомата.

Здесь **<statements>** – описания переходов и вложенных автоматов.

- **final state <name> {<no statements>}**

Конечное состояние автомата.

- **on <event name> if (this.a.x1( )) execute this.a.z1( ), execute this.a.z2( ) transite to <new state name>;**

Эта конструкция задает переход в состояние **<new state name>** при событии **<event name>** и входной переменной **x1.**

Отдельно стоит сказать про конструкцию **`this.a.*( )`**. Предполагается, что в области видимости автомата определен объект **`a`**, в котором реализованы входные и выходные переменные. При этом **`this.a.*( )`** есть обращение к ним.

4. Создание автомата в среде *MPS*

Для реализации автомата в среде *MPS* необходимо создать проект и подключить к нему текстовый язык автоматного программирования. После этого в проекте можно создавать новые модели с автоматами внутри.

На рис. 1 показан открытый проект *vehicles* (транспортные средства) с подключенным языком автоматного программирования (s*tateMachine).*

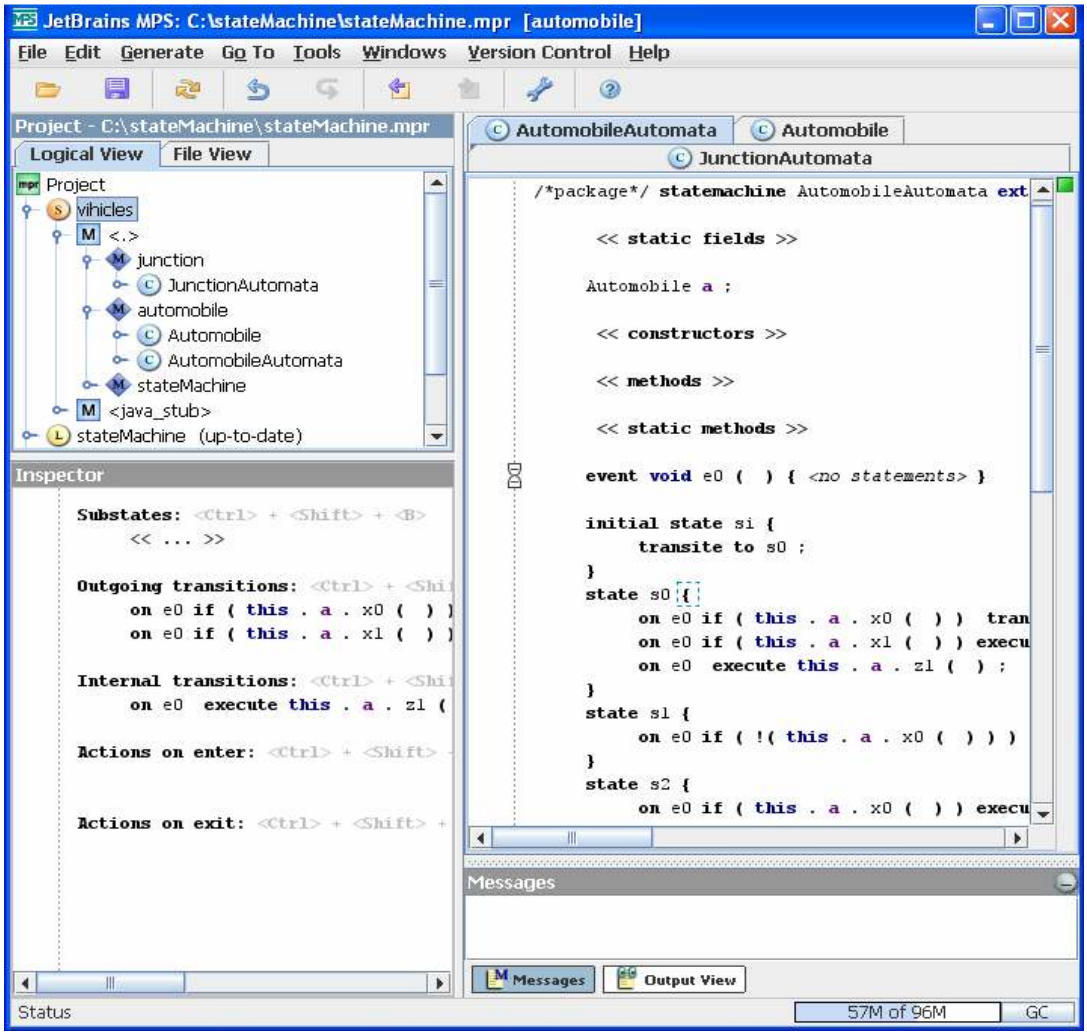

Рис. 1. Создание автомата в среде *MPS*

Моделями являются *junction* (перекресток) и *automobile*. При использовании рассматриваемого языка автомат создается заполнением некоторых шаблонов. Например, в состояние можно добавить конструкцию **“on * if * execute * transite to *“** и заполнить соответствующие элементы этого шаблона.

Подробно о создании моделей с помощью *MPS* можно прочесть в работе [5].

5. Структура программы

Поскольку текстовый язык автоматного программирования интегрируется с языком *Java*, то неавтоматная часть проекта “Мультиагентная система дорожного движения” первоначально созданная на *C++*, была переписана на *Java*.

Проект реализован как апплет.

Классы можно разделить на две категории: автоматные и вспомогательные.

Автоматные классы:

- `Automobile - класс автомобиль;`
- `AutomobileAutomata – автомат, управляющий автомобилем;`
- `AutomobileAutomataStates – состояния автомата, управляющего автомобилем;`
- `AutomobileAutomataByHand – реализация автомата, управляющего автомобилем, с помощью SWITCH-технологии;`
- `Junction - класс перекресток;`
- `JunctionAutomata – автомат, управляющий перекрестком;`
- `JunctionAutomataStates – состояния автомата, управляющего перекрестком;`
- `JunctionAutomata – автомат, управляющий перекрестком.`

Вспомогательные классы:

- `CWorld – класс карты;`
- `MainFrame – апплет;`

Автоматные классы используются для управления машинами и светофорами. Они приведены в приложениях.

Вспомогательные классы используются для обеспечения работы окна, реакции на действия пользователя, отображения карты и машин. Код вспомогательных классов в работе не приводится. Он опубликован на сайте http://is.ifmo.ru в разделе “Автоматный язык”.

6. Автомат `Automobile (A1)`

6.1. Описание

Данный автомат предназначен для управления автомобилем. Алгоритм управления следующий. Исходным является состояние 0 – "*Едем*". В данном состоянии машина перемещается по дороге. Если впереди оказывается другая машина, то автомат переходит в состояние 1 – "*Ждем машину*", и ждет до тех пор, пока она не уедет. За некоторое расстояние до перекрестка автомат переходит в состояние 2 – "*Едем перед перекрестком*", случайным образом выбирается направление поворота, и включаются поворотные огни. Если впереди оказывается другая машина, то автомат переходит в состояние 3 – "*Ждем машину перед перекрестком*", и ждет, пока она не уедет. Так же возможна остановка перед перекрестком, если он занят (состояние 4 – "*Стоим на перекрестке*"). Если перекресток свободен, то мы поворачиваем, выбрав состояние соответственно направлению поворота. После поворота автомат возвращается в исходное состояние 0.

6.2. События

`e0` – прошло время `dt`.

6.3. Входные переменные

`x0` – впереди стоит машина.

`x1` – впереди перекресток.

`x2` – перекресток (стоп линия).

`x3` – направление поворота.

`x4` – перекресток свободен.

`x5` – зеленый сигнал светофора.

`x6` – проезд перекрестка закончен.

6.4. Выходные воздействия

z0 – включить (выключить) фары.

z1 – проехать dx.

z2 – повернуть на dx.

z3 – задать направление поворота.

z4 – занять перекресток.

z5 – начать поворот.

z6 – освободить перекресток.

6.5. Схема связей автомата *A1*

Схема связей автомата *A1* приведена на рис. 2.

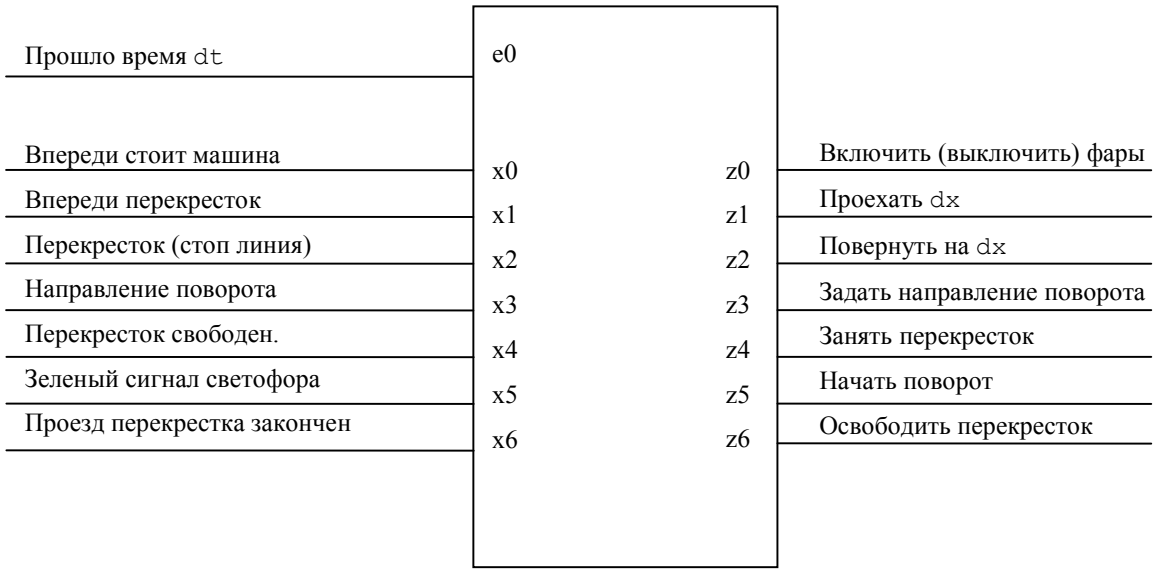

Рис. 2. Схема связей автомата *A1*

6.6. Граф переходов автомата *A1*

Граф переходов автомата *A1* приведен на рис. 3.

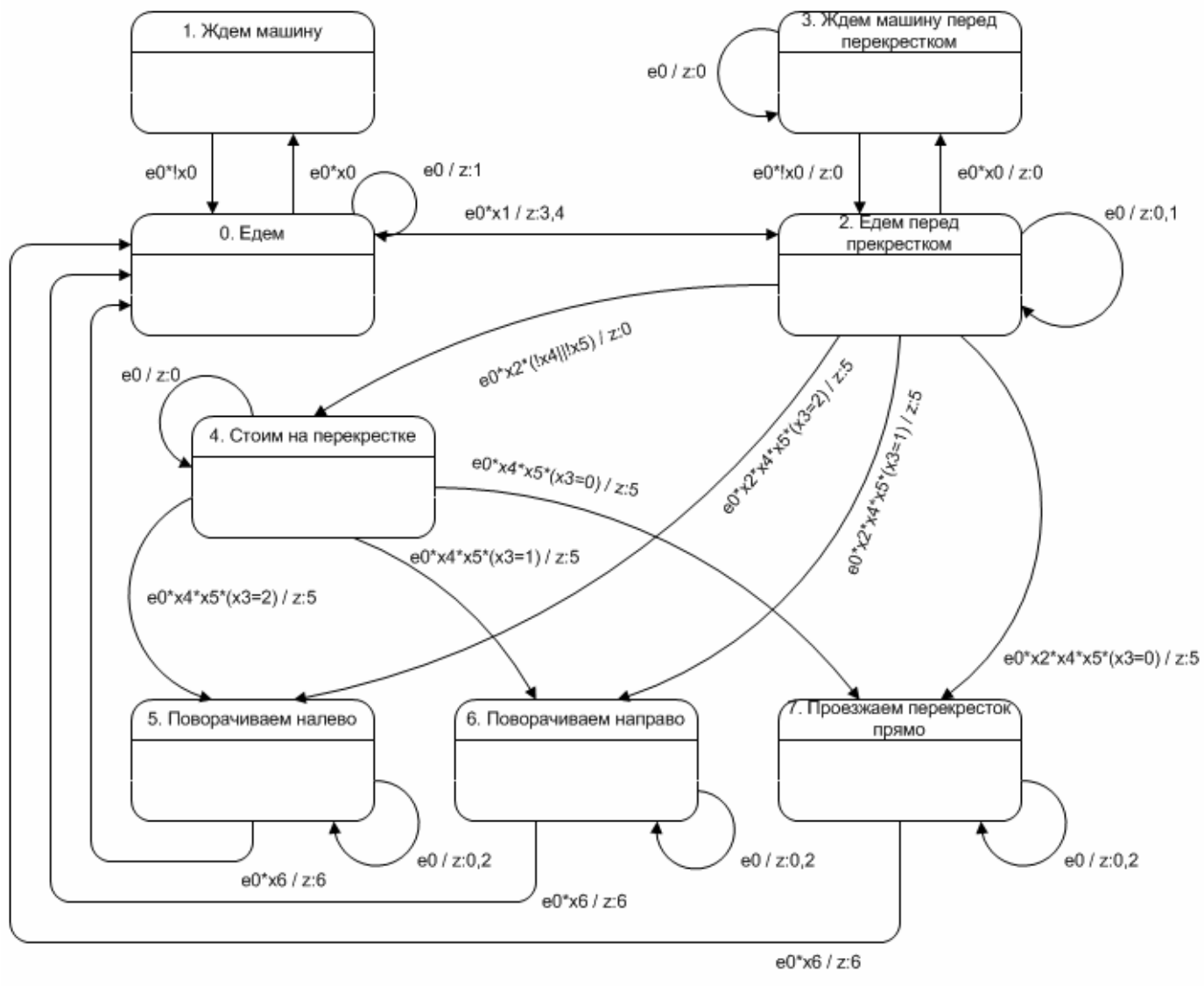

Рис. 3. Граф переходов автомата *A1*

На этом графе символ ‘*’ – конъюнкция, а, например, предикат (x3 = 1) следует понимать, как предикат, формирующий единицу при равенстве x3 единице и ноль в противном случае.

6.7. Реализация автомата *A1*

В данной работе представлено две реализации автомата *A1* – на языке *Java* и на текстовом языке автоматного программирования.

Реализация на языке *Java* написана по *SWITCH*-технологии (Приложение 1).

Описание автомата на текстовом языке автоматного программирования имеет вид:

```
event void e0 ( ) {<no statements>}

initial state si {
    transite to s0;
}
state s0 {
    on e0 if (this.a.x0( )) transite to s1;
    on e0 if (this.a.x1( )) execute this.a.z3( ), execute this.a.z4( )
             transite to s2;
    on e0 execute this.a.z1( );
}
state s1 {
    on e0 if (!(this.a.x0( ))) transite to s0;
}
state s2 {
    on e0 if (this.a.x0( )) execute this.a.z0( ) transite to s1;
    on e0 if (this.a.x2( ) && (!(this.a.x4( ))) || (!(this.a.x5( ))))
              this.a.z0( ) transite to s4;
    on e0 if (this.a.x2( ) && (this.a.x4( )) && (this.a.x5( )) &&
             (this.a.x3( ) == 2)) this.a.z5( ) transite to s5;
    on e0 if (this.a.x2( ) && (this.a.x4( )) && (this.a.x5( )) &&
             (this.a.x3( ) == 1)) this.a.z5( ) transite to s6;
    on e0 if (this.a.x2( ) && (this.a.x4( )) && (this.a.x5( )) &&
             (this.a.x3( ) == 0)) this.a.z5( ) transite to s7;
}
state s5 {
    on e0 if (this.a.x6( )) execute this.a.z6( ) transite to s0;
    on e0 execute this.a.z0( ), execute this.a.z2( );
}
state s6 {
    on e0 if (this.a.x6( )) execute this.a.z6( ) transite to s0;
    on e0 execute this.a.z0( ), execute this.a.z2( );
}
```

```
state s7 {
    on e0 if (this.a.x6( )) execute this.a.z6( ) transite to s0;
    on e0 execute this.a.z0( ), execute this.a.z2( );
}
```

Это описание приведено также в Приложении 2. Оно было написано в среде *MPS*.

После этого по нему были автоматически сгенерированы классы `AutomobileAutomata.java` и `AutomobileAutomataStates.java` (Приложение 3).

Какая реализация будет использоваться, зависит от значения флага `useAutoLang`.

В Приложении 4 можно найти программный код класса `Automobile.java`. В нем реализованы все входные и выходные воздействия.

7. Автомат `Junction (A2)`

7.1. Описание

Данный автомат предназначен для управления перекрестком. Один раз за время `dt` он обрабатывает событие `e0`, переключая текущий сигнал светофора.

7.2. События

`e0` – прошло время `dt`.

7.3. Схема связей автомата *A2*

Схема связей автомата *A2* приведена на рис. 4.

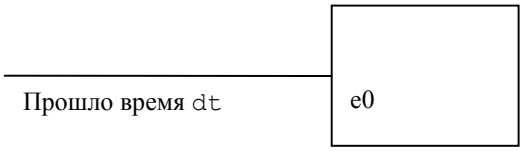

Рис. 4. Схема связей автомата *A2*

7.4. Граф переходов автомата *A2*

Граф переходов автомата *A2* приведен на рис. 5.

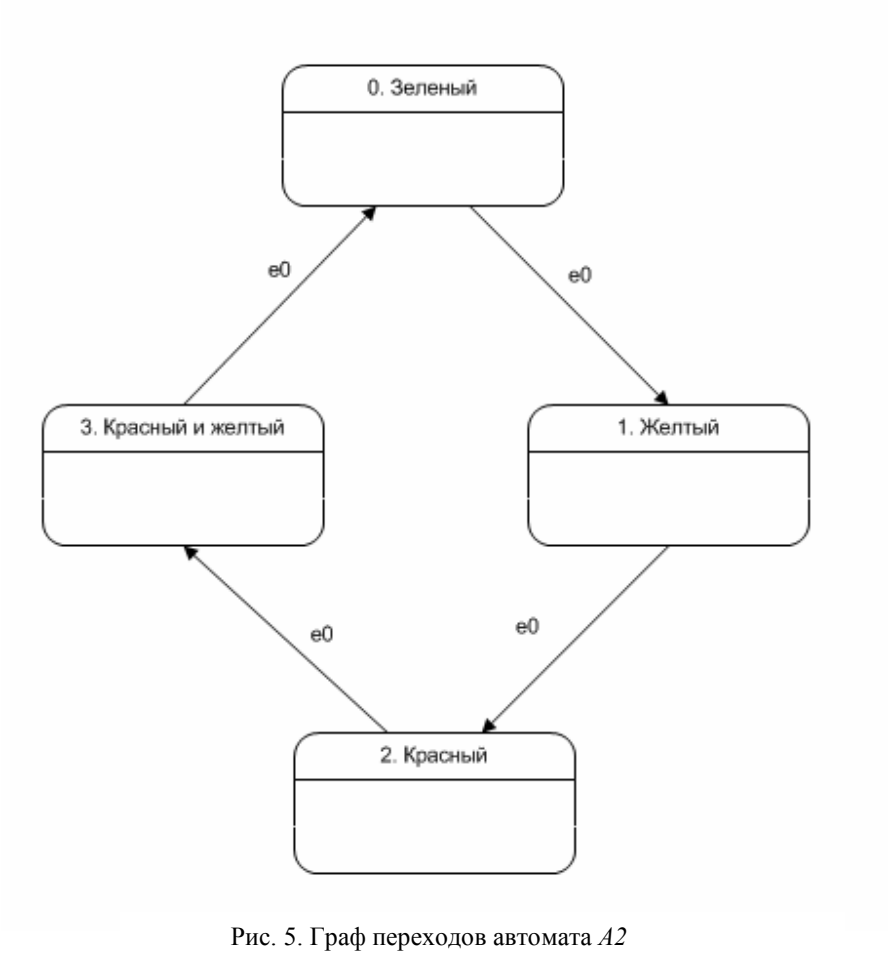

Рис. 5. Граф переходов автомата *A2*

7.5. Реализация автомата *A2*

С реализациями автомата *A2* все аналогично реализациям автомата *A1*.

Реализация на языке *Java* приведена в Приложении 5.

Описание автомата на текстовом языке автоматного программирования приведено в Приложении 6, а автоматически сгенерированые классы `JunctionAutomata.java` и `JunctionAutomataStates.java` - в Приложении 7.

Какая реализация будет использоваться, зависит от значения флага `useAutoLang`.

В Приложении 8 приведен класс `Junction.java` с реализацией входных и выходных воздействий.

8. Взаимодействие автоматов

Как отмечено выше, в программе имеется класс `Automobile`, основной частью которого является автомат управления *A1*. Каждый объект-автомобиль является экземпляром данного класса. При этом все автомобили (агенты) имеют одинаковую модель поведения, которая задается автоматом *A1*.

Каждый автомобиль работает в отдельном потоке.

В любой момент времени в зависимости от внешних условий автомобиль может находиться в одном из перечисленных восьми состояний.

Автоматы, управляющие машинами, в данной работе взаимодействуют не непосредственно, а с помощью меток (значений выходных функций), “выставляемых” на перекрестках.

Каждая машина, подъезжая к перекрестку, ставит метку в соответствующем массиве класса `CJunction`. После этого машина, подъехав к перекрестку, проверяет, занят он или нет – необходимо ли кого-либо попускать согласно правилам дорожного движения? В зависимости от занятости перекрестка, автомобиль проезжает его или ждет, когда перекресток освободится. Проехав его, машина снимает метку о себе, и тем самым, освобождает перекресток.

9. Описание интерфейса

Программа имеет интерфейс, показанный на рис. 6.

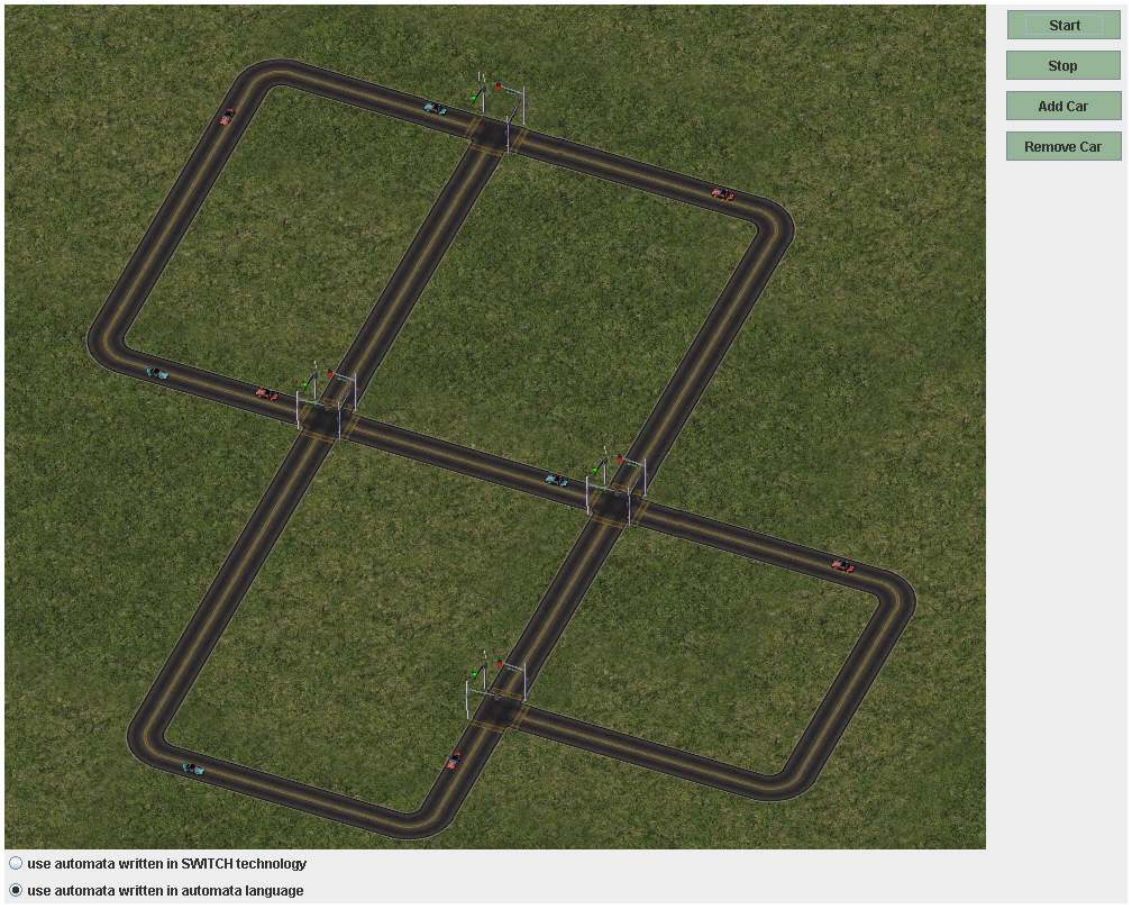

Рис. 6. Интерфейс пользователя

Нажатием на соответствующие кнопки панели управления можно запускать и приостанавливать процесс движения машин. При помощи мыши можно добавлять и удалять автомобили. Добавляемые автомобили появляются сразу после ближайшего перекрестка. Удалять можно только автомобили, находящиеся в состоянии один.

Также в нижней панели можно выбирать, какая реализация автомата будет использоваться: на языке автоматного программирования или сделанная при помощи *SWITCH*-технологии. Естественно, от этого выбора ничего не должно изменяться, так как они формально и изоморфно реализуют один и тот же автомат.

Заключение

Использование текстового языка автоматного программирования позволило изоморфно описывать графы переходов.

Недостатками текущей реализации языка являются относительная трудоемкость заполнения шаблонов и отсутствие возможности копирования фрагментов описания.

На сайте http://is.ifmo.ru/ в разделе “Автоматный язык” можно найти приложение и его исходные коды.

Ссылку на язык, аннотация

Источники

1. *Красильников Н. Н., Шалыто А. А.* Мультиагентная система дорожного движения. СПбГУ ИТМО. 2005. http://is.ifmo.ru/projects/vehicles/.

2. *Туккель Н. И., Шалыто А.А.* Switch-технология – автоматный подход к созданию программного обеспечения "реактивных" систем. http://is.ifmo.ru/works/switch/1 .

3. *Шалыто А. А.* Switch-технология. Алгоритмизация и программирование задач логического управления. СПб.: Наука, 1998. http://is.ifmo.ru/books/alg_log.

4. *Инструментальное средство Unimod*. http://unimod.sourceforge.net/intro.html

5. *MPS от JetBrains*. http://www.jetbrains.com/mps/.

6. *Дмитриев С.* Языково-ориентированное программирование: следующая парадигма. RSDN Magazine. 2005. № 5. http://www.rsdn.ru/article/philosophy/LOP.xml

7. *Фаулер М.* Языковой инструментарий: новая жизнь языков предметной области (Domain Specific Languages) //Клиент-сервер. 2005. № 3. http://www.optim.su/cs/2005/3/fowler/fowler.asp

Приложение 1. Реализация автомата *A1* на языке *Java*

```
void Main(int e) {
    switch (a.y0) {
        case 0:// Едем
            if ((e == 0) && a.x0()) {
                a.y0 = 1;
            } else if ((e == 0) && a.x1()) {
                a.z3();
                a.z4();
                a.y0 = 2;
            } else if ((e == 0)) {
                a.z1();
            }
            break;

        case 1:// Ждем машину
            if ((e == 0) && !a.x0()) {
                a.y0 = 0;
            }
            break;

        case 2:// Едем перед  прекрестком
            if ((e == 0) && a.x0()) {
                a.z0();
                a.y0 = 3;
            } else if ((e == 0) && a.x2() && (!a.x4() || !a.x5())) {
                a.z0();
                a.y0 = 4;
            } else if ((e == 0) && a.x2() && a.x4()
                        && a.x5() && (a.x3() == 0)) {
                a.z5();
                a.y0 = 7;
            } else if ((e == 0) && a.x2() && a.x4()
                        && a.x5() && (a.x3() == 1)) {
                a.z5();
                a.y0 = 6;
            } else if ((e == 0) && a.x2() && a.x4()
                        && a.x5() && (a.x3() == 2)) {
                a.z5();
                a.y0 = 5;
            } else if ((e == 0)) {
                a.z0();
                a.z1();
            }
            break;

        case 3:// Ждем машину пред перекрестком
            if ((e == 0) && !a.x0()) {
                a.z0();
                a.y0 = 2;
            } else if ((e == 0)) {
                a.z0();
            }
            break;
```

```
        case 4:// Стоим на перекрестке
            if ((e == 0) && a.x4() && a.x5() && (a.x3() == 0)) {
                a.z5();
                a.y0 = 7;
            } else if ((e == 0) && a.x4() && a.x5() && (a.x3() == 2)) {
                a.z5();
                a.y0 = 5;
            } else if ((e == 0) && a.x4() && a.x5() && (a.x3() == 1)) {
                a.z5();
                a.y0 = 6;
            } else if ((e == 0)) {
                a.z0();
            }
            break;

        case 5:// Поворачиваем налево
            if ((e == 0) && a.x6()) {
                a.z6();
                a.y0 = 0;
            } else if ((e == 0)) {
                a.z0();
                a.z2();
            }
            break;

        case 6:// Поворачиваем направо
            if ((e == 0) && a.x6()) {
                a.z6();
                a.y0 = 0;
            } else if ((e == 0)) {
                a.z0();
                a.z2();
            }
            break;

        case 7:// Проезжаем перекресток прямо
            if ((e == 0) && a.x6()) {
                a.z6();
                a.y0 = 0;
            } else if ((e == 0)) {
                a.z0();
                a.z2();
            }
            break;

        default:

    }
}
```

Приложение 2. Реализация автомата *A1* на текстовом языке автоматного программирования

```
/*package*/ statemachine AutomobileAutomata extends <none> implements <none>{

      << static fields >>

      Automobile a;

      << constructors >>
      << methods >>
      << static methods >>

      event void e0 ( ) {<no statements>}

      initial state si {
          transite to s0;
      }
      state s0 {
          on e0 if (this.a.x0( )) transite to s1;
          on e0 if (this.a.x1( )) execute this.a.z3( ), execute this.a.z4( )
                    transite to s2;
          on e0 execute this.a.z1( );
      }
      state s1 {
          on e0 if (!(this.a.x0( ))) transite to s0;
      }
      state s2 {
          on e0 if (this.a.x0( )) execute this.a.z0( ) transite to s1;
          on e0 if (this.a.x2( ) && (!(this.a.x4( ))) || (!(this.a.x5( ))))
                     this.a.z0( ) transite to s4;
          on e0 if (this.a.x2( ) && (this.a.x4( )) && (this.a.x5( )) &&
                    (this.a.x3( ) == 2)) this.a.z5( ) transite to s5;
          on e0 if (this.a.x2( ) && (this.a.x4( )) && (this.a.x5( )) &&
                    (this.a.x3( ) == 1)) this.a.z5( ) transite to s6;
          on e0 if (this.a.x2( ) && (this.a.x4( )) && (this.a.x5( )) &&
                    (this.a.x3( ) == 0)) this.a.z5( ) transite to s7;
      }
      state s5 {
```

```
        on e0 if (this.a.x6( )) execute this.a.z6( ) transite to s0;
        on e0 execute this.a.z0( ), execute this.a.z2( );
    }
    state s6 {
        on e0 if (this.a.x6( )) execute this.a.z6( ) transite to s0;
        on e0 execute this.a.z0( ), execute this.a.z2( );
    }
    state s7 {
        on e0 if (this.a.x6( )) execute this.a.z6( ) transite to s0;
        on e0 execute this.a.z0( ), execute this.a.z2( );
    }
}
```

Приложение 3. Код автомата *A1* на *Java*, сгенерированный средой *MPS* по текстовому описанию

AutomobileAutomata.java

```
package Vehicles;
/*Generated by MPS 09.01.2007 20:55:51 */

import java.util.Stack;

public class AutomobileAutomata {

  public Automobile a;
  public AutomobileAutomataStates myState;

  public  AutomobileAutomata() {
    this.transiteToStable(AutomobileAutomataStates.ROOT);
  }

  public AutomobileAutomataStates getState() {
    return this.myState;
  }
  public void setState(AutomobileAutomataStates state) {
    this.myState = state;
  }
  public void enterState(AutomobileAutomataStates state) {
    if(state == AutomobileAutomataStates.s0) {
    }
    if(state == AutomobileAutomataStates.s1) {
    }
    if(state == AutomobileAutomataStates.s2) {
    }
    if(state == AutomobileAutomataStates.s3) {
    }
    if(state == AutomobileAutomataStates.s4) {
    }
    if(state == AutomobileAutomataStates.s5) {
    }
    if(state == AutomobileAutomataStates.s6) {
    }
    if(state == AutomobileAutomataStates.s7) {
    }
  }
  public void enterStateRange(AutomobileAutomataStates fromSuperstateExclusive,
              AutomobileAutomataStates toSubstateInclusive) {
    Stack<AutomobileAutomataStates> stack = new Stack<AutomobileAutomataStates>();
    for(AutomobileAutomataStates curState = toSubstateInclusive ;
                  curState != fromSuperstateExclusive ;
                  curState = curState.superstate) {
      stack.push(curState);
    }
    while(!(stack.isEmpty())) {
      AutomobileAutomataStates curState = stack.pop();
      this.enterState(curState);
    }
  }
  public AutomobileAutomataStates transiteToStable(AutomobileAutomataStates state) {
    while(state.initialSubstate != null) {
      this.transiteFromInitial(state.initialSubstate);
```

```
      state = state.initialSubstate.getPointedState();
      this.enterState(state);
    }
    return state;
  }
  public void transiteFromInitial(AutomobileAutomataStates state) {
    if(state == AutomobileAutomataStates.si) {
    }
  }
  public void exitState(AutomobileAutomataStates state) {
    if(state == AutomobileAutomataStates.s0) {
    }
    if(state == AutomobileAutomataStates.s1) {
    }
    if(state == AutomobileAutomataStates.s2) {
    }
    if(state == AutomobileAutomataStates.s3) {
    }
    if(state == AutomobileAutomataStates.s4) {
    }
    if(state == AutomobileAutomataStates.s5) {
    }
    if(state == AutomobileAutomataStates.s6) {
    }
    if(state == AutomobileAutomataStates.s7) {
    }
  }
  public void exitStateRange(AutomobileAutomataStates fromSubstateInclusive,
AutomobileAutomataStates toSuperstateExclusive) {
    for(AutomobileAutomataStates curState = fromSubstateInclusive ;
                curState != toSuperstateExclusive ;
                curState = curState.superstate) {
      this.exitState(curState);
    }
  }
  public void e0() {
    for(AutomobileAutomataStates sourceState = this.getState() ;
                sourceState != AutomobileAutomataStates.ROOT ;
                 sourceState = sourceState.superstate) {
      if(((sourceState == AutomobileAutomataStates.s0) && (this.a.x0())) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s0, AutomobileAutomataStates.ROOT);
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s1);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s1));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s0) && (this.a.x1())) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s0, AutomobileAutomataStates.ROOT);
        this.a.z3();
        this.a.z4();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s2);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s2));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s1) && (!(this.a.x0()))) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s1, AutomobileAutomataStates.ROOT);
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s0);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s0));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s2) && (this.a.x0())) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s2, AutomobileAutomataStates.ROOT);
        this.a.z0();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s3);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s3));
        break;
      }
```

```
      if(((sourceState == AutomobileAutomataStates.s2) && (this.a.x2() && (!(this.a.x4())
|| !(this.a.x5()))))) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s2, AutomobileAutomataStates.ROOT);
        this.a.z0();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s4);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s4));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s2) && (this.a.x2() && this.a.x4() &&
                  this.a.x5() && (this.a.x3() == 2))) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s2, AutomobileAutomataStates.ROOT);
        this.a.z5();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s5);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s5));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s2) && (this.a.x2() && this.a.x4() &&
                  this.a.x5() && (this.a.x3() == 1))) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s2, AutomobileAutomataStates.ROOT);
        this.a.z5();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s6);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s6));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s2) && (this.a.x2() && this.a.x4() &&
                  this.a.x5() && (this.a.x3() == 0))) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s2, AutomobileAutomataStates.ROOT);
        this.a.z5();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s7);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s7));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s3) && (!(this.a.x0())))) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s3, AutomobileAutomataStates.ROOT);
        this.a.z0();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s2);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s2));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s4) && (this.a.x4() && this.a.x5() &&
                (this.a.x3() == 2))) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s4, AutomobileAutomataStates.ROOT);
        this.a.z5();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s5);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s5));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s4) && (this.a.x4() && this.a.x5() &&
                (this.a.x3() == 1))) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s4, AutomobileAutomataStates.ROOT);
        this.a.z5();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s6);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s6));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s4) && (this.a.x4() && this.a.x5() &&
                (this.a.x3() == 0))) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s4, AutomobileAutomataStates.ROOT);
        this.a.z5();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s7);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s7));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s5) && (this.a.x6())) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s5, AutomobileAutomataStates.ROOT);
        this.a.z6();
```

```
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s0);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s0));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s6) && (this.a.x6())) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s6, AutomobileAutomataStates.ROOT);
        this.a.z6();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s0);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s0));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s7) && (this.a.x6())) && (true)) {
        this.exitStateRange(AutomobileAutomataStates.s7, AutomobileAutomataStates.ROOT);
        this.a.z6();
        this.enterStateRange(AutomobileAutomataStates.ROOT, AutomobileAutomataStates.s0);
        this.setState(this.transiteToStable(AutomobileAutomataStates.s0));
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s0) && (true)) && (true)) {
        this.a.z1();
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s2) && (true)) && (true)) {
        this.a.z0();
        this.a.z1();
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s3) && (true)) && (true)) {
        this.a.z0();
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s4) && (true)) && (true)) {
        this.a.z0();
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s5) && (true)) && (true)) {
        this.a.z0();
        this.a.z2();
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s6) && (true)) && (true)) {
        this.a.z0();
        this.a.z2();
        break;
      }
      if(((sourceState == AutomobileAutomataStates.s7) && (true)) && (true)) {
        this.a.z0();
        this.a.z2();
        break;
      }
    }
    return;
  }
}
```

AutomobileAutomataStates.java

```
package Vehicles;
/*Generated by MPS 09.01.2007 20:55:51 */


public enum AutomobileAutomataStates {
  ROOT(),
  si(AutomobileAutomataStates.ROOT, true, false),
  s0(AutomobileAutomataStates.ROOT, false, false),
  s1(AutomobileAutomataStates.ROOT, false, false),
  s2(AutomobileAutomataStates.ROOT, false, false),
  s3(AutomobileAutomataStates.ROOT, false, false),
  s4(AutomobileAutomataStates.ROOT, false, false),
  s5(AutomobileAutomataStates.ROOT, false, false),
  s6(AutomobileAutomataStates.ROOT, false, false),
  s7(AutomobileAutomataStates.ROOT, false, false);

  public AutomobileAutomataStates superstate;
  public AutomobileAutomataStates initialSubstate;
  public AutomobileAutomataStates pointedState;
  public boolean isFinal;
  public boolean isInitial;

 AutomobileAutomataStates() {
    this.isInitial = false;
    this.isFinal = false;
    this.superstate = null;
  }
 AutomobileAutomataStates(AutomobileAutomataStates superstate, boolean isInitial,
        boolean isFinal) {
    this.superstate = superstate;
    this.isInitial = isInitial;
    this.isFinal = isFinal;
    if(this.isInitial) {
      this.superstate.initialSubstate = this;
    }
  }

  public static void initPointedState() {
    AutomobileAutomataStates.si.pointedState = AutomobileAutomataStates.s0;
  }

  public AutomobileAutomataStates getPointedState() {
    if(this.isInitial) {
      if(this.pointedState == null) {
        AutomobileAutomataStates.initPointedState();
      }
      return this.pointedState;
    } else
    {
      return null;
    }
  }
}
```

Приложение 4. Класс `Automobile`

```
package Vehicles;

import java.awt.*;

/**
 * @author Nick
 */

public class Automobile implements Runnable {
    boolean useAutoLang = true;

    // Состояния
    final static int AUTOMOBILE_STATE_RUN = 0;
    final static int AUTOMOBILE_STATE_WAIT_CAR = 1;
    final static int AUTOMOBILE_STATE_NEAR_JUNC = 2;
    final static int AUTOMOBILE_STATE_WAIT_CAR_NEAR_JUNC = 3;
    final static int AUTOMOBILE_STATE_WAIT_ON_JANC = 4;
    final static int AUTOMOBILE_STATE_TURN_LEFT = 5;
    final static int AUTOMOBILE_STATE_TURN_RIGHT = 6;
    final static int AUTOMOBILE_STATE_TURN_THROUGH = 7;

    // События
    final static int EVENT_CLOCK = 0;

    // Направления
    final static int DIRECTION_N = 0;
    final static int DIRECTION_E = 1;
    final static int DIRECTION_S = 2;
    final static int DIRECTION_W = 3;

    final static int DIRECTION_THROUGH = 0;
    final static int DIRECTION_RIGHT = 1;
    final static int DIRECTION_LEFT = 2;

    int y0;

    int automobileModel;

    boolean lights;

    int velociti;

    int startJunc;
    int endJunc;

    int oldDirection;

    int currentJunc;
    int turnDirection;
    int turnPosition;

    int distance;

    boolean stop;

    Point point = new Point();

    Junction[] juncs;
    Automobile[] cars;

    AutomobileAutomata auto = new AutomobileAutomata();

    AutomobileAutomataByHand autoByHand = new AutomobileAutomataByHand();
```

```
private Thread thread = null;

Automobile() {
    auto.a = this;
    autoByHand.a = this;

    auto.setState(auto.myState.s0);
}

void Start() {
    if (thread == null) {
        stop = false;

        thread = new Thread(this);

        thread.start();
    }
}

void Stop() {
    stop = true;
    thread = null;
}

boolean x0() // Впереди стоит машина
{
    boolean i = IsCarNear();
    return i;
}

boolean x1() // Впереди перекресток
{
    boolean i = IsJunctionNear();
    return i;
}

boolean x2() // Перекресток (стоп линия)
{
    boolean i = IsJunction();
    return i;
}

int x3() // Направление поворота
{
    int i = turnDirection;
    return i;
}

boolean x4() // Перекресток свободен
{
    boolean i = IsJuncFree();
    return i;
}

boolean x5() // Зеленый сигнал светофора
{
    boolean i = IsGreenLight();
    return i;
}

boolean x6() // Проезд перекрестка закончен
{
    boolean i = true;

    if (y0 == AUTOMOBILE_STATE_TURN_THROUGH)
```

```
        i = (turnPosition > 10);
    if (y0 == AUTOMOBILE_STATE_TURN_LEFT)
        i = (turnPosition > 10);
    if (y0 == AUTOMOBILE_STATE_TURN_RIGHT)
        i = (turnPosition > 6);

    return i;
}

void z0() // Включить (выключить) фары
{
    lights = !lights;
}

void z1() // Проехать dx
{
    Run();
}

void z2() // Повернуть на dx
{
    turnPosition++;
}

void z3() // Задать направление поворота
{
    turnDirection = GenerateTurnDirection();
}

void z4() // Занять перекресток
{
    currentJunc = endJunc;

    oldDirection = GetDirection();

    if (turnDirection == DIRECTION_THROUGH)
        juncs[currentJunc].carsRunThrough[GetDirection()]++;
    if (turnDirection == DIRECTION_RIGHT)
        juncs[currentJunc].carsTurnRight[GetDirection()]++;
    if (turnDirection == DIRECTION_LEFT)
        juncs[currentJunc].carsTurnLeft[GetDirection()]++;
}

void z5() // Начать поворот
{
    turnPosition = 0;

    if (turnDirection == DIRECTION_THROUGH)
        y0 = AUTOMOBILE_STATE_TURN_THROUGH;
    if (turnDirection == DIRECTION_RIGHT)
        y0 = AUTOMOBILE_STATE_TURN_RIGHT;
    if (turnDirection == DIRECTION_LEFT)
        y0 = AUTOMOBILE_STATE_TURN_LEFT;

    SetDirection(turnDirection);
}

void z6() // Освободить перекресток
{
    if (turnDirection == DIRECTION_THROUGH)
        juncs[currentJunc].carsRunThrough[oldDirection]--;
    if (turnDirection == DIRECTION_RIGHT)
        juncs[currentJunc].carsTurnRight[oldDirection]--;
    if (turnDirection == DIRECTION_LEFT)
        juncs[currentJunc].carsTurnLeft[oldDirection]--;
}
```

```
////////////////////////////////////////////////////////////////////////////
//
//
//

    void Run() {
        distance += velociti;
    }

    int GetDirection() {
        if ((juncs[endJunc].x - juncs[startJunc].x) >= 0 &&
            (juncs[endJunc].y - juncs[startJunc].y) >= 0) {
            return DIRECTION_E;
        }
        if ((juncs[endJunc].x - juncs[startJunc].x) >= 0 &&
            (juncs[endJunc].y - juncs[startJunc].y) < 0) {
            return DIRECTION_N;
        }
        if ((juncs[endJunc].x - juncs[startJunc].x) < 0 &&
            (juncs[endJunc].y - juncs[startJunc].y) >= 0) {
            return DIRECTION_S;
        }
        if ((juncs[endJunc].x - juncs[startJunc].x) < 0 &&
            (juncs[endJunc].y - juncs[startJunc].y) < 0) {
            return DIRECTION_W;
        }

        return 0;
    }


    int GetDirection(int sj, int ej) {
        if ((juncs[ej].x - juncs[sj].x) >= 0 && (juncs[ej].y - juncs[sj].y) >= 0) {
            return DIRECTION_E;
        }
        if ((juncs[ej].x - juncs[sj].x) >= 0 && (juncs[ej].y - juncs[sj].y) < 0) {
            return DIRECTION_N;
        }
        if ((juncs[ej].x - juncs[sj].x) < 0 && (juncs[ej].y - juncs[sj].y) >= 0) {
            return DIRECTION_S;
        }
        if ((juncs[ej].x - juncs[sj].x) < 0 && (juncs[ej].y - juncs[sj].y) < 0) {
            return DIRECTION_W;
        }

        return 0;
    }

    int GenerateTurnDirection() {
        while (true) {
            int random = (int) (Math.random() * 3.5);

            int oldEndJunc = endJunc;
            int oldStartJunc = startJunc;

            if ((juncs[oldEndJunc].neighbours[random] != -1) &&
                (juncs[oldEndJunc].neighbours[random] != oldStartJunc)) {
                int oldDirection = GetDirection();
                int newDirection = GetDirection(endJunc,
                                    juncs[oldEndJunc].neighbours[random]);

                if (oldDirection == DIRECTION_N) {
                    if (newDirection == DIRECTION_N)
                        return DIRECTION_THROUGH;
                    if (newDirection == DIRECTION_E)
```

```
                    return DIRECTION_RIGHT;
                if (newDirection == DIRECTION_S)
                    return -1;
                if (newDirection == DIRECTION_W)
                    return DIRECTION_LEFT;
            }
            if (oldDirection == DIRECTION_E) {
                if (newDirection == DIRECTION_N)
                    return DIRECTION_LEFT;
                if (newDirection == DIRECTION_E)
                    return DIRECTION_THROUGH;
                if (newDirection == DIRECTION_S)
                    return DIRECTION_RIGHT;
                if (newDirection == DIRECTION_W)
                    return -1;
            }
            if (oldDirection == DIRECTION_S) {
                if (newDirection == DIRECTION_N)
                    return -1;
                if (newDirection == DIRECTION_E)
                    return DIRECTION_LEFT;
                if (newDirection == DIRECTION_S)
                    return DIRECTION_THROUGH;
                if (newDirection == DIRECTION_W)
                    return DIRECTION_RIGHT;
            }
            if (oldDirection == DIRECTION_W) {
                if (newDirection == DIRECTION_N)
                    return DIRECTION_RIGHT;
                if (newDirection == DIRECTION_E)
                    return -1;
                if (newDirection == DIRECTION_S)
                    return DIRECTION_LEFT;
                if (newDirection == DIRECTION_W)
                    return DIRECTION_THROUGH;
            }

            break;
        }
    }

    return DIRECTION_THROUGH;
}

void SetDirection(int td) {
    int direction = GetDirection();
    int newDirection = DIRECTION_N;

    if (direction == DIRECTION_N) {
        if (td == DIRECTION_THROUGH)
            newDirection = DIRECTION_N;
        if (td == DIRECTION_RIGHT)
            newDirection = DIRECTION_E;
        if (td == DIRECTION_LEFT)
            newDirection = DIRECTION_W;
    }
    if (direction == DIRECTION_E) {
        if (td == DIRECTION_THROUGH)
            newDirection = DIRECTION_E;
        if (td == DIRECTION_RIGHT)
            newDirection = DIRECTION_S;
        if (td == DIRECTION_LEFT)
            newDirection = DIRECTION_N;
    }
    if (direction == DIRECTION_S) {
        if (td == DIRECTION_THROUGH)
```

```
            newDirection = DIRECTION_S;
        if (td == DIRECTION_RIGHT)
            newDirection = DIRECTION_W;
        if (td == DIRECTION_LEFT)
            newDirection = DIRECTION_E;
    }
    if (direction == DIRECTION_W) {
        if (td == DIRECTION_THROUGH)
            newDirection = DIRECTION_W;
        if (td == DIRECTION_RIGHT)
            newDirection = DIRECTION_N;
        if (td == DIRECTION_LEFT)
            newDirection = DIRECTION_S;
    }

    distance = 37;

    startJunc = endJunc;
    endJunc = juncs[endJunc].neighbours[newDirection];
}

int GetNextJunc(int td) {
    int direction = GetDirection();
    int newDirection = DIRECTION_N;

    if (direction == DIRECTION_N) {
        if (td == DIRECTION_THROUGH)
            newDirection = DIRECTION_N;
        if (td == DIRECTION_RIGHT)
            newDirection = DIRECTION_E;
        if (td == DIRECTION_LEFT)
            newDirection = DIRECTION_W;
    }
    if (direction == DIRECTION_E) {
        if (td == DIRECTION_THROUGH)
            newDirection = DIRECTION_E;
        if (td == DIRECTION_RIGHT)
            newDirection = DIRECTION_S;
        if (td == DIRECTION_LEFT)
            newDirection = DIRECTION_N;
    }
    if (direction == DIRECTION_S) {
        if (td == DIRECTION_THROUGH)
            newDirection = DIRECTION_S;
        if (td == DIRECTION_RIGHT)
            newDirection = DIRECTION_W;
        if (td == DIRECTION_LEFT)
            newDirection = DIRECTION_E;
    }
    if (direction == DIRECTION_W) {
        if (td == DIRECTION_THROUGH)
            newDirection = DIRECTION_W;
        if (td == DIRECTION_RIGHT)
            newDirection = DIRECTION_N;
        if (td == DIRECTION_LEFT)
            newDirection = DIRECTION_S;
    }

    return juncs[endJunc].neighbours[newDirection];
}

boolean IsCarNear() {
    for (Automobile car : cars) {
        if (car != this) {
            if ((car.startJunc == startJunc) && (car.endJunc == endJunc) &&
                    ((car.distance - distance) < 60) &&
```

```
                        ((car.distance - distance) > 0)) {
                    return true;
                }
            }
        }

        return false;
    }

    boolean IsJunction() {
        int distanceBetweenJuncs = (int) Math.sqrt(Math.pow(juncs[endJunc].y -
juncs[startJunc].y, 2.0) + Math.pow(juncs[endJunc].x - juncs[startJunc].x, 2.0));

        if ((distanceBetweenJuncs - distance) <= 25) {
            distance = distanceBetweenJuncs - 25;
            return true;
        }

        return false;
    }

    boolean IsJunctionNear() {
        int distanceBetweenJuncs = (int) Math.sqrt(Math.pow(juncs[endJunc].y -
juncs[startJunc].y, 2.0) + Math.pow(juncs[endJunc].x - juncs[startJunc].x, 2.0));

        return (distanceBetweenJuncs - distance) < 60;
    }

    int GetTurnFromDirection(int d) {
        return 0;
    }

    boolean IsGreenLight() {
        int direction = GetDirection();

        return ((juncs[currentJunc].type == Junction.JUNC_TYPE_NO_TL) ||
                ((juncs[currentJunc].y1 == Junction.JUNC_STATE_GREEN_NS) && (direction ==
DIRECTION_N || direction == DIRECTION_S)) ||
                ((juncs[currentJunc].y1 == Junction.JUNC_STATE_GREEN_EW) && (direction ==
DIRECTION_W || direction == DIRECTION_E)));
    }

    boolean IsJuncFree() {
        int direction = GetDirection();

        if (juncs[currentJunc].carsRunThrough[direction] +
                juncs[currentJunc].carsTurnLeft[direction] +
                juncs[currentJunc].carsTurnRight[direction] != 1)
            return false;

        for (Automobile car : cars) {
            if (car != this) {
                if ((car.startJunc == currentJunc) && (car.endJunc ==
GetNextJunc(turnDirection)) &&
                        ((car.distance) < 60)) {
                    return false;
                }
            }
        }

        if (direction == DIRECTION_N) {
            if (turnDirection == DIRECTION_LEFT)
                if ((juncs[currentJunc].carsRunThrough[DIRECTION_S] != 0) ||
                        (juncs[currentJunc].carsTurnRight[DIRECTION_S] != 0))
                    return false;
        }
```

```
        if (direction == DIRECTION_E) {
            if (turnDirection == DIRECTION_LEFT)
                if ((juncs[currentJunc].carsRunThrough[DIRECTION_W] != 0) ||
                        (juncs[currentJunc].carsTurnRight[DIRECTION_W] != 0))
                    return false;
        }
        if (direction == DIRECTION_S) {
            if (turnDirection == DIRECTION_LEFT)
                if ((juncs[currentJunc].carsRunThrough[DIRECTION_N] != 0) ||
                        (juncs[currentJunc].carsTurnRight[DIRECTION_N] != 0))
                    return false;
        }
        if (direction == DIRECTION_W) {
            if (turnDirection == DIRECTION_LEFT)
                if ((juncs[currentJunc].carsRunThrough[DIRECTION_E] != 0) ||
                        (juncs[currentJunc].carsTurnRight[DIRECTION_E] != 0))
                    return false;
        }

        return true;
    }

    public synchronized void run() {
        while (!stop) {
            if (useAutoLang) {
                auto.e0();

                if (auto.getState() == AutomobileAutomataStates.s0) y0 = 0;
                if (auto.getState() == AutomobileAutomataStates.s1) y0 = 1;
                if (auto.getState() == AutomobileAutomataStates.s2) y0 = 2;
                if (auto.getState() == AutomobileAutomataStates.s3) y0 = 3;
                if (auto.getState() == AutomobileAutomataStates.s4) y0 = 4;
                if (auto.getState() == AutomobileAutomataStates.s5) y0 = 5;
                if (auto.getState() == AutomobileAutomataStates.s6) y0 = 6;
                if (auto.getState() == AutomobileAutomataStates.s7) y0 = 7;
            } else {
                autoByHand.Main(0);

                if (y0 == 0) auto.setState(AutomobileAutomataStates.s0);
                if (y0 == 1) auto.setState(AutomobileAutomataStates.s1);
                if (y0 == 2) auto.setState(AutomobileAutomataStates.s2);
                if (y0 == 3) auto.setState(AutomobileAutomataStates.s3);
                if (y0 == 4) auto.setState(AutomobileAutomataStates.s4);
                if (y0 == 5) auto.setState(AutomobileAutomataStates.s5);
                if (y0 == 6) auto.setState(AutomobileAutomataStates.s6);
                if (y0 == 7) auto.setState(AutomobileAutomataStates.s7);
            }

            try {
                Thread.sleep((long) (30));
            }
            catch (InterruptedException e) {
                System.out.println("InterruptedException in Vehicles.Automobile");
            }
        }
    }
}
```

Приложение 5. Реализация автомата *A2* на языке *Java*

```
void Main(int e) {
    switch (j.y1) {
        case 0:// Зеленый
            if ((e == 0)) {
                j.y1 = 1;
            }
            break;

        case 1:// Желтый
            if ((e == 0)) {
                j.y1 = 2;
            }
            break;

        case 2:// Красный
            if ((e == 0)) {
                j.y1 = 3;
            }
            break;

        case 3:// Красный и желтый
            if ((e == 0)) {
                j.y1 = 0;
            }
            break;
    }
}
```

Приложение 6. Реализация автомата *A2* на текстовом языке автоматного программирования

```
/*package*/ statemachine AutomobileAutomata extends <none> implements <none>{
      << static fields >>
      << fields >>
      << constructors >>
      << methods >>
      << static methods >>

      event void e0 ( ) {<no statements>}

      initial state si {
          transite to s0;
      }
      state s0 {
          on e0 transite to s1;
      }
      state s1{
          on e0 transite to s2
      }
      state s2{
          on e0 transite to s3
      }
      state s3{
          on e0 transite to s0
      }
}
```

Приложение 7. Код автомата *A2* на *Java*, сгенерированный средой *MPS* по текстовому описанию.

JunctionAutomata.java

```
package Vehicles;
/*Generated by MPS 09.01.2007 20:57:58 */

import java.util.Stack;

public class JunctionAutomata {

  public JunctionAutomataStates myState;

  public  JunctionAutomata() {
    this.transiteToStable(JunctionAutomataStates.ROOT);
  }

  public JunctionAutomataStates getState() {
    return this.myState;
  }
  public void setState(JunctionAutomataStates state) {
    this.myState = state;
  }
  public void enterState(JunctionAutomataStates state) {
    if(state == JunctionAutomataStates.s0) {
    }
    if(state == JunctionAutomataStates.s1) {
    }
    if(state == JunctionAutomataStates.s2) {
    }
    if(state == JunctionAutomataStates.s3) {
    }
  }
  public void enterStateRange(JunctionAutomataStates fromSuperstateExclusive,
                              JunctionAutomataStates toSubstateInclusive) {
    Stack<JunctionAutomataStates> stack = new Stack<JunctionAutomataStates>();
    for(JunctionAutomataStates curState = toSubstateInclusive ;
                    curState != fromSuperstateExclusive ;
                    curState = curState.superstate) {
      stack.push(curState);
```

```
  }
  while(!(stack.isEmpty())) {
    JunctionAutomataStates curState = stack.pop();
    this.enterState(curState);
  }
}
public JunctionAutomataStates transiteToStable(JunctionAutomataStates state) {
  while(state.initialSubstate != null) {
    this.transiteFromInitial(state.initialSubstate);
    state = state.initialSubstate.getPointedState();
    this.enterState(state);
  }
  return state;
}
public void transiteFromInitial(JunctionAutomataStates state) {
  if(state == JunctionAutomataStates.si) {
  }
}
public void exitState(JunctionAutomataStates state) {
  if(state == JunctionAutomataStates.s0) {
  }
  if(state == JunctionAutomataStates.s1) {
  }
  if(state == JunctionAutomataStates.s2) {
  }
  if(state == JunctionAutomataStates.s3) {
  }
}
public void exitStateRange(JunctionAutomataStates fromSubstateInclusive,
                           JunctionAutomataStates toSuperstateExclusive) {
  for(JunctionAutomataStates curState = fromSubstateInclusive ;
                 curState != toSuperstateExclusive ;
                 curState = curState.superstate) {
    this.exitState(curState);
  }
}
public void e0() {
  for(JunctionAutomataStates sourceState = this.getState() ;
              sourceState != JunctionAutomataStates.ROOT ;
              sourceState = sourceState.superstate) {
    if(((sourceState == JunctionAutomataStates.s0) && (true)) && (true)) {
      this.exitStateRange(JunctionAutomataStates.s0, JunctionAutomataStates.ROOT);
      this.enterStateRange(JunctionAutomataStates.ROOT, JunctionAutomataStates.s1);
      this.setState(this.transiteToStable(JunctionAutomataStates.s1));
```

```
        break;
      }
      if(((sourceState == JunctionAutomataStates.s1) && (true)) && (true)) {
        this.exitStateRange(JunctionAutomataStates.s1, JunctionAutomataStates.ROOT);
        this.enterStateRange(JunctionAutomataStates.ROOT, JunctionAutomataStates.s2);
        this.setState(this.transiteToStable(JunctionAutomataStates.s2));
        break;
      }
      if(((sourceState == JunctionAutomataStates.s2) && (true)) && (true)) {
        this.exitStateRange(JunctionAutomataStates.s2, JunctionAutomataStates.ROOT);
        this.enterStateRange(JunctionAutomataStates.ROOT, JunctionAutomataStates.s3);
        this.setState(this.transiteToStable(JunctionAutomataStates.s3));
        break;
      }
      if(((sourceState == JunctionAutomataStates.s3) && (true)) && (true)) {
        this.exitStateRange(JunctionAutomataStates.s3, JunctionAutomataStates.ROOT);
        this.enterStateRange(JunctionAutomataStates.ROOT, JunctionAutomataStates.s0);
        this.setState(this.transiteToStable(JunctionAutomataStates.s0));
        break;
      }
    }
    return;
  }
}
```

JunctionAutomataStates.java

```
package Vehicles;
/*Generated by MPS 09.01.2007 20:57:59 */


public enum JunctionAutomataStates {
  ROOT(),
  si(JunctionAutomataStates.ROOT, true, false),
  s0(JunctionAutomataStates.ROOT, false, false),
  s1(JunctionAutomataStates.ROOT, false, false),
  s2(JunctionAutomataStates.ROOT, false, false),
  s3(JunctionAutomataStates.ROOT, false, false);

  public JunctionAutomataStates superstate;
  public JunctionAutomataStates initialSubstate;
  public JunctionAutomataStates pointedState;
  public boolean isFinal;
  public boolean isInitial;

 JunctionAutomataStates() {
    this.isInitial = false;
    this.isFinal = false;
    this.superstate = null;
  }
 JunctionAutomataStates(JunctionAutomataStates superstate, boolean isInitial,
```

```
                    boolean isFinal) {
    this.superstate = superstate;
    this.isInitial = isInitial;
    this.isFinal = isFinal;
    if(this.isInitial) {
      this.superstate.initialSubstate = this;
    }
  }

  public static void initPointedState() {
    JunctionAutomataStates.si.pointedState = JunctionAutomataStates.s0;
  }

  public JunctionAutomataStates getPointedState() {
    if(this.isInitial) {
      if(this.pointedState == null) {
        JunctionAutomataStates.initPointedState();
      }
      return this.pointedState;
    } else
    {
      return null;
    }
  }
}
```

Приложение 8. Класс `Junction`

```
package Vehicles;

/**
 * @author Nick
 */

public class Junction implements Runnable {
    boolean useAutoLang = true;

    // Состояния
    final static int JUNC_STATE_RED = 2;
    final static int JUNC_STATE_YELLOW_TO_RED = 1;
    final static int JUNC_STATE_YELLOW_TO_GREEN = 3;
    final static int JUNC_STATE_GREEN = 0;

    // Копии названий состояний
    final static int JUNC_STATE_RED_NS = 2;
    final static int JUNC_STATE_YELLOW_TO_RED_NS = 1;
    final static int JUNC_STATE_YELLOW_TO_GREEN_NS = 3;
    final static int JUNC_STATE_GREEN_NS = 0;

    final static int JUNC_STATE_RED_EW = 0;
    final static int JUNC_STATE_YELLOW_TO_RED_EW = 3;
    final static int JUNC_STATE_YELLOW_TO_GREEN_EW = 1;
    final static int JUNC_STATE_GREEN_EW = 2;

    // События
    final static int EVENT_CLOCK = 0;

    // Типы перекрестков
    final static int JUNC_TYPE_NO_TL = -1;
    final static int JUNC_TYPE_X = 0;
    final static int JUNC_TYPE_T_N = 1;
    final static int JUNC_TYPE_T_W = 2;
    final static int JUNC_TYPE_T_S = 3;
    final static int JUNC_TYPE_T_E = 4;

    int y1;

    int x, y;
    int neighbours[] = new int[4];

    int type;

    int intervalRed;
    int intervalGreen;
    int intervalYellowToRed;
    int intervalYellowToGreen;

    int carsTurnLeft[] = new int[4];
    int carsTurnRight[] = new int[4];
    int carsRunThrough[] = new int[4];


    private Thread thread;

    JunctionAutomata auto = new JunctionAutomata();
    JunctionAutomataByHand autoByHand = new JunctionAutomataByHand();

    Junction() {
        autoByHand.j = this;

        thread = new Thread(this);
```

```
        for (int i = 0; i < 4; i++) {
            carsTurnLeft[i] = 0;
            carsTurnRight[i] = 0;
            carsRunThrough[i] = 0;
        }

        auto.setState(auto.myState.s0);

        y1 = JUNC_STATE_GREEN;
    }

    void Start() {
        thread.start();
    }

    public synchronized void run() {
        while (true) {

            if (useAutoLang) {

                auto.e0();

                if (auto.getState() == auto.myState.s0) y1 = 0;
                if (auto.getState() == auto.myState.s1) y1 = 1;
                if (auto.getState() == auto.myState.s2) y1 = 2;
                if (auto.getState() == auto.myState.s3) y1 = 3;

            } else {
                autoByHand.Main(0);
            }

            try {
                if (y1 == JUNC_STATE_RED)
                    Thread.sleep(intervalRed);
                if (y1 == JUNC_STATE_GREEN)
                    Thread.sleep(intervalGreen);
                if (y1 == JUNC_STATE_YELLOW_TO_GREEN)
                    Thread.sleep(intervalYellowToGreen);
                if (y1 == JUNC_STATE_YELLOW_TO_RED)
                    Thread.sleep(intervalYellowToRed);
            }
            catch (InterruptedException e) {
            }
        }
    }
}
```