Санкт-Петербургский государственный университет
информационных технологий, механики и оптики

Кафедра “Компьютерные технологии”

Н.Н. Красильников, А.А. Шалыто

Мультиагентная система дорожного движения

Программирование с явным выделением состояний
Проектная документация

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

Санкт-Петербург
2005

Оглавление

Введение

В настоящее время в мире большое внимание уделяется исследованиям в области разработки мультиагентных систем [1]. В целях изучения подобных систем, автоматного программирования и *switch*-технологии [2, 3] в данной работе строится система дорожного движения автомобилей.

Основным этапом создания программы является ее проектирование. На этом этапе строятся управляющие автоматы, и определяется их взаимодействие. Далее с помощью инструментального средства *Visio2Switch* [4] по построенным автоматов генерируется код на *C++*. Этот код «изоморфен» графам переходов используемых автоматов.

1. Постановка задачи

Цель работы – создание мультиагентной системы дорожного движения автомобилей и демонстрация ее работы.

Автомобили должны 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, разделенные двойной разделительной полосой и все машины движутся с одной скоростью.

2. Описание интерфейса

Программа имеет стандартный интерфейс *Windows* приложения (рис. 1).

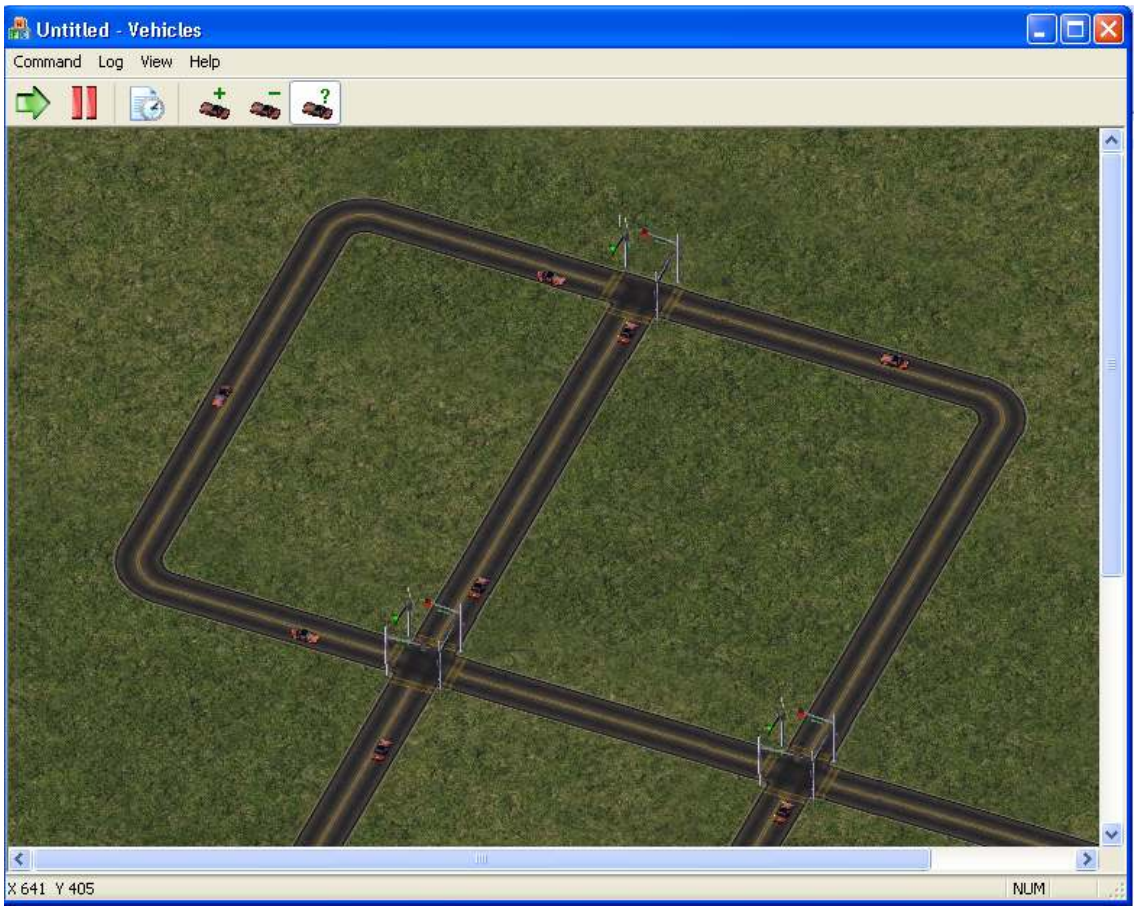

Рис.1. Интерфейс пользователя

Выбором пунктов меню или нажатием на соответствующие кнопки панели инструментов можно запускать и приостанавливать процесс движения машин, смотреть протоколы движения, для каждого автомобиля, добавлять и удалять автомобили.

3. Структура программы

Классы можно разделить на две категории: вспомогательные и автоматные.

Автоматные классы:

- `CAutomobile - класс автомобиль;`
- `CJunction - класс перекресток.`

Вспомогательные классы:

- `CWorld – класс карты;`
- `CLogDialog – класс диалога протоколирования;`
- `CMainFrame – класс рамки окна;`
- `CVehiclesApp – класс приложения;`
- `CVehiclesDoc – класс документа;`
- `CVehiclesView – класс отображения;`
- `CAboutDlg – класс справки.`

Автоматные классы используются для управления машинами и светофорами и для протоколирования и 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.

4. Класс `CAutomobile`

4.1. Описание

Данный класс 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.

4.2. События

`e0` – прошло время `dt`.

4.3. 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.

4.4. 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

4.5. Схема связей автомата *A1*

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

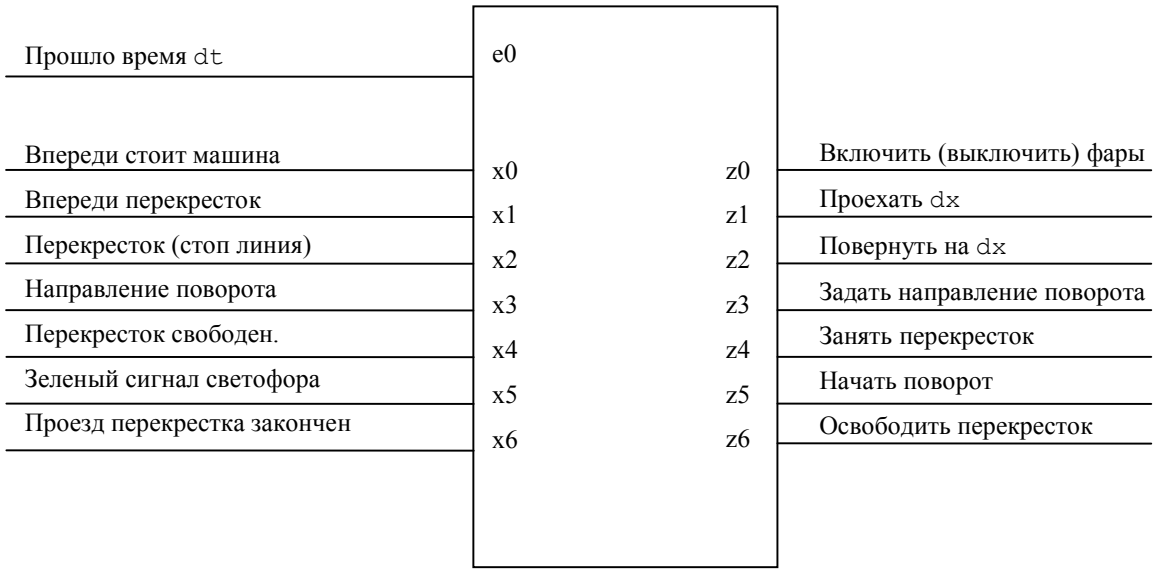

Рис.2. Схема связей автомата *A1*

4.6. Граф переходов автомата *A1*

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

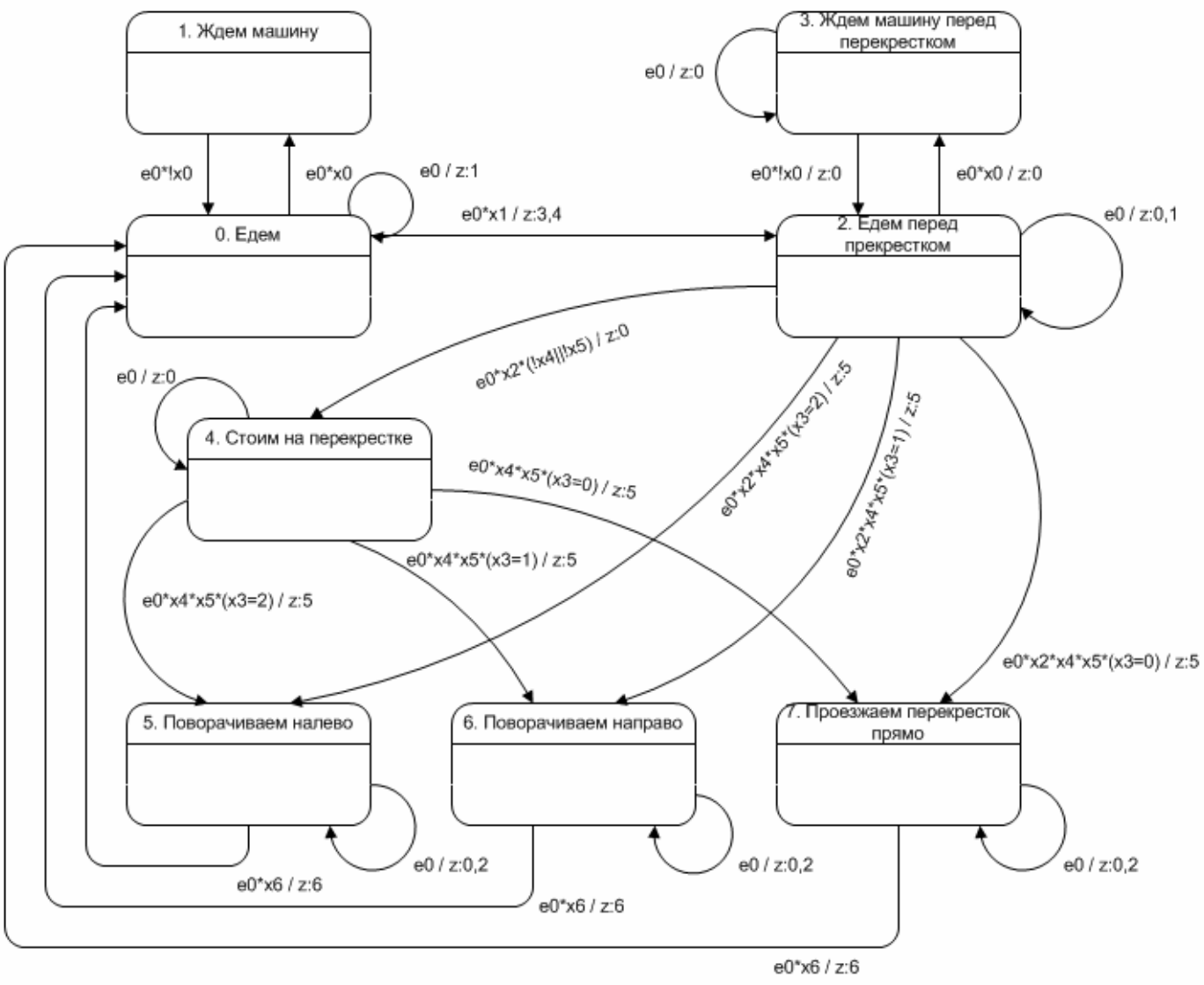

Рис.3. Граф переходов автомата *A1*

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.

5. Класс `CJunction`

5.1. Описание

Данный класс предназначен для управления перекрестком. Раз в промежуток времени `dt` он обрабатывает событие `e0`, просто переключая текущий сигнал светофора.

5.2. События

`e0` – прошло время `dt`.

5.3. Схема связей автомата *A2*

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

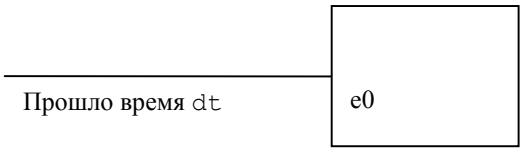

Рис.4. Схема связей автомата *A2*

5.4. Граф переходов автомата *A2*

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

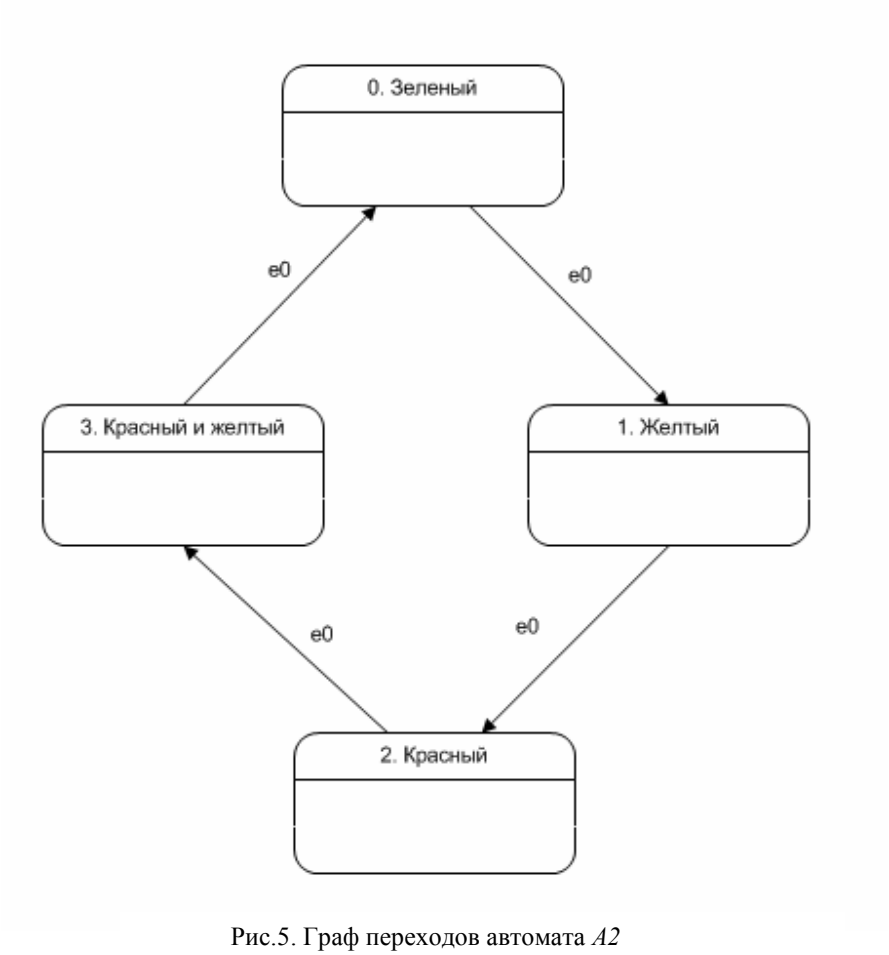

Рис.5. Граф переходов автомата *A2*

6. Взаимодействие автоматов

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

В каждый момент времени в зависимости от внешних условий автомобиль может находиться в одном из перечисленных в разд. 4.5 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-нибудь 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.

7. Протоколирование

Для каждой машины в программе ведется протокол ее состояний, проверяемых условий и действий.

Приведем в качестве примера фрагмент такого протокола:

```
Автомобиль: завершил обработку события e0 (Прошло время dt) в состоянии 2 (Едем перед перекрестком)
Автомобиль: в состоянии 2 (Едем перед перекрестком) запущен с событием e0 (Прошло время dt)
    x0 - Впереди стоит машина - вернул 0
    x2 - Перекресток (стоп линия) - вернул 1
    x4 - Перекресток свободен - вернул 1
    x5 - Зеленый сигнал светофора - вернул 1
    x2 - Перекресток (стоп линия) - вернул 1
    x4 - Перекресток свободен - вернул 1
    x5 - Зеленый сигнал светофора - вернул 1
    x3 - Направление поворота - вернул 2
    x2 - Перекресток (стоп линия) - вернул 1
    x4 - Перекресток свободен - вернул 1
    x5 - Зеленый сигнал светофора - вернул 1
    x3 - Направление поворота - вернул 2
    x2 - Перекресток (стоп линия) - вернул 1
    x4 - Перекресток свободен - вернул 1
    x5 - Зеленый сигнал светофора - вернул 1
    x3 - Направление поворота - вернул 2
        z5 - Начать поворот
Автомобиль: завершил обработку события e0 (Прошло время dt) в состоянии 5 (Поворачиваем налево)
Автомобиль: в состоянии 5 (Поворачиваем налево) запущен с событием e0 (Прошло время dt)
    x6 - Проезд перекрестка закончен - вернул 0
        z0 - Включить (выключить) фары
        z2 - Повернуть на dx
Автомобиль: завершил обработку события e0 (Прошло время dt) в состоянии 5 (Поворачиваем налево)
```

На рис. 6 фрагмент протокола для машины № 8.

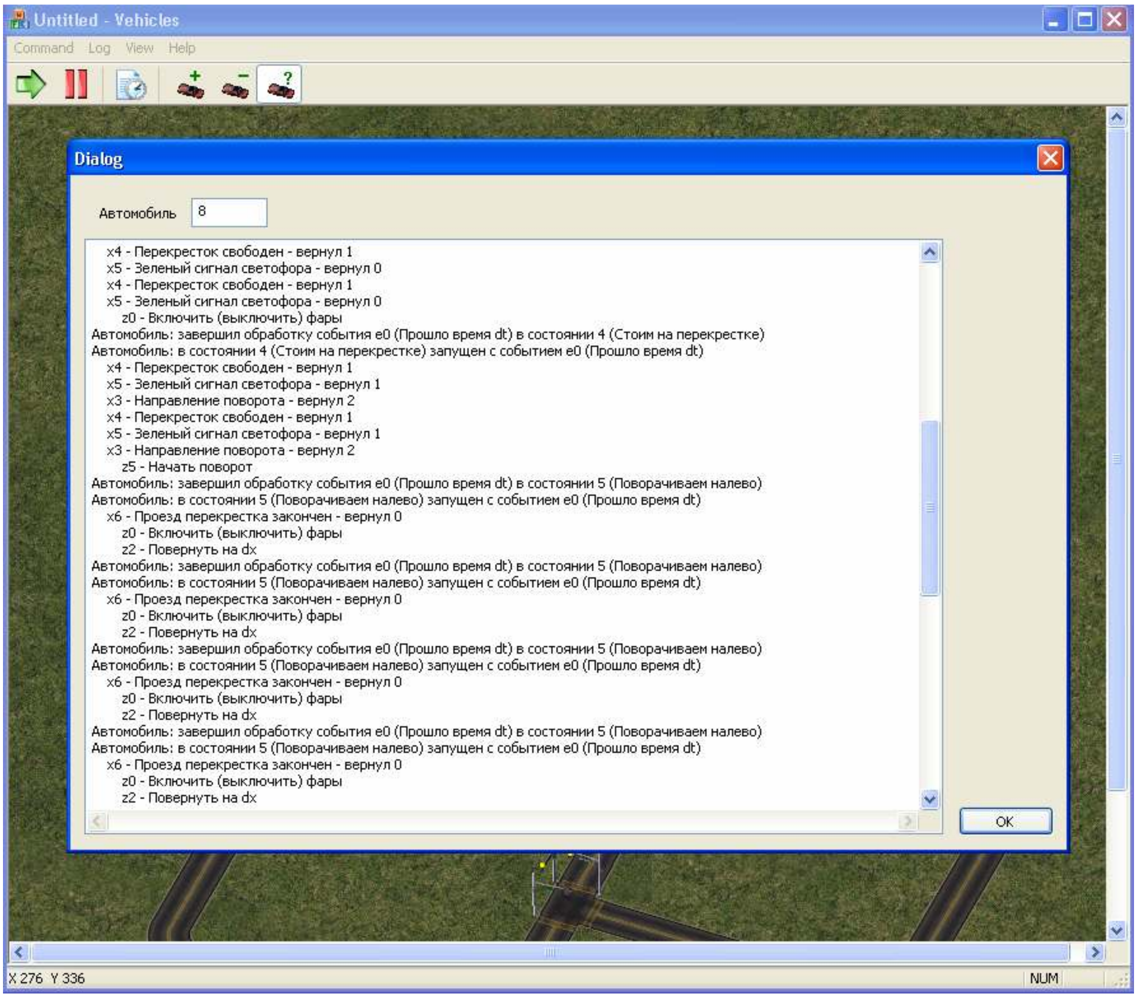

Рис.6. Протокол

Заключение

Автоматное программирование позволило довольно просто и наглядно решить поставленную задачу, а инструментальное средство *Visio2SWITCH* – автоматически реализовать изображенные в *Microsoft Visio* автоматы на языке *C++*. Это позволило избежать ошибок при реализации.

На сайте http://is.ifmo.ru/ в разделе “Проекты” размещено приложение и его исходные коды.

Источники

1. *Naumov L., Korneev G., Shalyto A.* Methods of Object-Oriented Reactive Agents Implementation on the Basis of Finite Automata //2005 International Conference on «Integration of Knowledge Intensive Multi-Agent Systems: Modeling, Exploration and Engineering». *KIMAS-05*. Boston: IEEE Boston Section. 2005, pp.460-465. http://is.ifmo.ru/articles_en/_kimas05-1.pdf

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.* Алгоритмизация и программирование задач логического управления СПбГУ ИТМО. 1998. http://is.ifmo.ru/books/alg_log

4. *Головешин А.* Конвертер для преобразования графов переходов в текст программы на языке *Си* (*Visio2Switch*). 2002. http://is.ifmo.ru/progeny/visio2switch/

Приложение 1. Реализация автомата *A1* на языке *C++*

```
void CAutomobile::Main(int e)
{
	int y_old = y0;

	log_a_begin(y_old, e);

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

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

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

```
            z5();
            y0 = 5;
      }
      else if((e == 0))
      {
            z0(); z1();
      }
      break;

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

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

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

case 6:// Поворачиваем направо
      if((e == 0) && x6())
      {
            z6();
```

```
                y0 = 0;
        }
        else if((e == 0))
        {
                z0(); z2();
        }
        break;

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

default:
        log_write('!',
                  "Ошибка в автомате A0: неизвестный номер состояния!");
        ;
}

log_a_end(y0, e);
}
```

Приложение 2. Реализация автомата *A2* на языке *C++*

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

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

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

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

Приложение 3. Класс CAutomobile

Automobile.h

```
#pragma once
#include "Junction.h"
#include "Vector"

// Состояния
#define AUTOMOBILE_STATE_RUN                        0
#define AUTOMOBILE_STATE_WAIT_CAR                   1
#define AUTOMOBILE_STATE_NEAR_JUNC                  2
#define AUTOMOBILE_STATE_WAIT_CAR_NEAR_JUNC         3
#define AUTOMOBILE_STATE_WAIT_ON_JANC               4
#define AUTOMOBILE_STATE_TURN_LEFT                  5
#define AUTOMOBILE_STATE_TURN_RIGHT                 6
#define AUTOMOBILE_STATE_TURN_THROUGH               7

// События
#define EVENT_CLOCK                                 0

// Направления
#define DIRECTION_N                                 0
#define DIRECTION_E                                 1
#define DIRECTION_S                                 2
#define DIRECTION_W                                 3

#define DIRECTION_THROUGH                           0
#define DIRECTION_RIGHT                             1
#define DIRECTION_LEFT                              2

class CAutomobile
{
public:
       int y0;

       bool x0(); // Впереди стоит машина
       bool x1(); // Впереди перекресток
       bool x2(); // Перекресток (стоп линия)
    int x3(); // Направление поворота
       bool x4(); // Перекресток свободен
       bool x5(); // Зеленый сигнал светофора
       bool x6(); // Проезд перекрестка закончен

       void z0(); // Включить (выключить) фары
       void z1(); // Проехать dx
       void z2(); // Повернуть на dx
       void z3(); // Задать направление поворота
       void z4(); // Занять перекресток
       void z5(); // Начать поворот
       void z6(); // Освободить перекресток

       CString log[100];
       int currentLogString;

       int automobileModel;

       bool lights;

       int velociti;

       int startJunc;
       int endJunc;
```

```
int oldDirection;

int currentJunc;
int turnDirection;
int turnPosition;

int distance;

CPoint point;

std::vector<CJunction> *juncs;
std::vector<CAutomobile> *cars;

CAutomobile(void);
~CAutomobile(void);

void Main(int event);

void Run();
bool IsCarNear();
bool IsJunction();
bool IsJunctionNear();
bool IsGreenLight();
bool IsJuncFree();

int GenerateTurnDirection();
void SetDirection(int td);
int GetDirection();
int GetDirection(int sj, int ej);
int GetTurnFromDirection(int d);


typedef struct{
	int	dig;
	const char* n;
	const char* n_name;
} int_str2_t;

typedef struct{
	const char* n;
	const char* n_name;
} str2_t;

typedef struct{
	const char* n;
	const char* n_name;
	str2_t* str2;
} str3_t;

int_str2_t e_str2[1];

str2_t x_str2[7];

str2_t z_str2[7];

str2_t a0_str2[8];

str3_t A_str3[1];

//--------------------------------------------------------------------------
void log_a_begin(int y, int e)
{
	log_a_begin_user(a0_str2[y].n, a0_str2[y].n_name);
}
//--------------------------------------------------------------------------
void log_a_end(int y, int e)
```

```
{
	log_a_end_user(a0_str2[y].n, a0_str2[y].n_name);
}
//---------------------------------------------------------------------------
void log_x(int x, int res)
{
	log_x_user(x_str2[x].n, x_str2[x].n_name, res);
}
//---------------------------------------------------------------------------
void log_z(int z)
{
	log_z_user(z_str2[z].n, z_str2[z].n_name);
}
//---------------------------------------------------------------------------
void log_write(char ch, const char* str)
{
	log_write_user(ch, str);
}
//---------------------------------------------------------------------------

//---------------------------------------------------------------------------
void log_a_begin_user(const char* y, const char* y_name)
{
	// LOG_GRAPH_BEGIN 'a'('a_name'): в состоянии 'y'() запущен с событием
	'e'('e_name')

	currentLogString = (currentLogString + 1)%100;

	log[currentLogString] = "Автомобиль: ";
	log[currentLogString] += "в состоянии ";
	log[currentLogString] += y;
	log[currentLogString] += " (";
	log[currentLogString] += y_name;
	log[currentLogString] += ") запущен с событием ";
	log[currentLogString] += "e0";
	log[currentLogString] += " (Прошло время dt)";
	log[currentLogString] += "\r\n";
}
void log_a_end_user(const char* y, const char* y_name)
{
	// LOG_GRAPH_END 'a'('a_name'): завершил обработку события 'e'('e_name') в
	состоянии 'y'()

	currentLogString = (currentLogString + 1)%100;

	log[currentLogString] = "Автомобиль: ";
	log[currentLogString] += "завершил обработку события ";
	log[currentLogString] += "e0";
	log[currentLogString] += " (Прошло время dt) ";
	log[currentLogString] += "в состоянии ";
	log[currentLogString] += y;
	log[currentLogString] += " (";
	log[currentLogString] += y_name;
	log[currentLogString] += ")";
	log[currentLogString] += "\r\n";
}
//---------------------------------------------------------------------------
void log_x_user(const char* x, const char* x_name, int res)
{
	// LOG_X 'x' - 'x_name' - вернул 'res'

	currentLogString = (currentLogString + 1)%100;

	log[currentLogString] = "    ";
	log[currentLogString] += x;
	log[currentLogString] += " - ";
```

```
            log[currentLogString] += x_name;
            log[currentLogString] += " - вернул ";
            log[currentLogString] += (char)(res + '0');
            log[currentLogString] += "\r\n";
      }
      //-------------------------------------------------------------------------
      void log_z_user(const char* z, const char* z_name)
      {
            // LOG_Z 'z' - 'z_name'

            currentLogString = (currentLogString + 1)%100;

            log[currentLogString] = "    ";
            log[currentLogString] += "    ";
            log[currentLogString] += z;
            log[currentLogString] += " - ";
            log[currentLogString] += z_name;
            log[currentLogString] += "\r\n";
      }
      //-------------------------------------------------------------------------
      void log_write_user(char ch, const char* str)
      {
            // ch, str

            currentLogString = (currentLogString + 1)%100;

            log[currentLogString] = ch;
            log[currentLogString] += " ";
            log[currentLogString] += str;
            log[currentLogString] += "\r\n";
      }
      //-------------------------------------------------------------------------
};
```

Automobile.cpp

```
#include "StdAfx.h"
#include "Resource.h"
#include "Math.h"
#include "List"
#include ".\automobile.h"

CAutomobile::CAutomobile(void)
{
      x_str2[0].n = "x0";
      x_str2[1].n = "x1";
      x_str2[2].n = "x2";
      x_str2[3].n = "x3";
      x_str2[4].n = "x4";
      x_str2[5].n = "x5";
      x_str2[6].n = "x6";

      x_str2[0].n_name = "Впереди стоит машина";
      x_str2[1].n_name = "Впереди перекресток";
      x_str2[2].n_name = "Перекресток (стоп линия)";
      x_str2[3].n_name = "Направление поворота";
      x_str2[4].n_name = "Перекресток свободен";
      x_str2[5].n_name = "Зеленый сигнал светофора";
      x_str2[6].n_name = "Проезд перекрестка закончен";

      z_str2[0].n = "z0";
      z_str2[1].n = "z1";
      z_str2[2].n = "z2";
      z_str2[3].n = "z3";
```

```
	z_str2[4].n = "z4";
	z_str2[5].n = "z5";
	z_str2[6].n = "z6";

	z_str2[0].n_name = "Включить (выключить) фары";
	z_str2[1].n_name = "Проехать dx";
	z_str2[2].n_name = "Повернуть на dx";
	z_str2[3].n_name = "Задать направление поворота";
	z_str2[4].n_name = "Занять перекресток";
	z_str2[5].n_name = "Начать поворот";
	z_str2[6].n_name = "Освободить перекресток";

	a0_str2[0].n = "0";
	a0_str2[1].n = "1";
	a0_str2[2].n = "2";
	a0_str2[3].n = "3";
	a0_str2[4].n = "4";
	a0_str2[5].n = "5";
	a0_str2[6].n = "6";
	a0_str2[7].n = "7";

	a0_str2[0].n_name = "Едем";
	a0_str2[1].n_name = "Ждем машину";
	a0_str2[2].n_name = "Едем перед перекрестком";
	a0_str2[3].n_name = "Ждем машину пред перекрестком";
	a0_str2[4].n_name = "Стоим на перекрестке";
	a0_str2[5].n_name = "Поворачиваем налево";
	a0_str2[6].n_name = "Поворачиваем направо";
	a0_str2[7].n_name = "Проезжаем перекресток прямо";

	currentLogString = 0;

}

CAutomobile::~CAutomobile(void)
{
}

//////////////////////////////////////////////////////////////////////////
//
// A0 - Автомобиль
//

void CAutomobile::Main(int e)
{
	int y_old = y0;

	log_a_begin(y_old, e);

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

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

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

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

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

```
                        y0 = 6;
                }
                else if((e == 0))
                {
                        z0();
                }
                break;

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

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

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

        default:
                log_write('!', "Ошибка в автомате A0: неизвестный номер состояния!");
                ;
        }

        log_a_end(y0, e);
}

bool CAutomobile::x0() // Впереди стоит машина
{
        int i = IsCarNear();
        log_x(0, i);
        return i;
}

bool CAutomobile::x1() // Впереди перекресток
{
        int i = IsJunctionNear();
        log_x(1, i);
        return i;
}
```

```
bool CAutomobile::x2() // Перекресток (стоп линия)
{
	int i = IsJunction();
	log_x(2, i);
	return i;
}

int CAutomobile::x3() // Направление поворота
{
	int i = turnDirection;
	log_x(3, i);
	return i;
}

bool CAutomobile::x4() // Перекресток свободен
{
	int i = IsJuncFree();
	log_x(4, i);
	return i;
}

bool CAutomobile::x5() // Зеленый сигнал светофора
{
	int i = IsGreenLight();
	log_x(5, i);
	return i;
}

bool CAutomobile::x6() // Проезд перекрестка закончен
{
	int i = 1;

	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);

	log_x(6, i);
	return i;
}

void CAutomobile::z0() // Включить (выключить) фары
{
	log_z(0);
	lights = !lights;
}

void CAutomobile::z1() // Проехать dx
{
	log_z(1);
	Run();
}

void CAutomobile::z2() // Повернуть на dx
{
	log_z(2);
	turnPosition++;
}

void CAutomobile::z3() // Задать направление поворота
{
	log_z(3);
	turnDirection = GenerateTurnDirection();
}
```

```
void CAutomobile::z4() // Занять перекресток
{
	log_z(4);
	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 CAutomobile::z5() // Начать поворот
{
	log_z(5);
	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 CAutomobile::z6() // Освободить перекресток
{
	log_z(6);
	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 CAutomobile::Run()
{
	distance += velociti;
}

int CAutomobile::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 CAutomobile::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 CAutomobile::GenerateTurnDirection()
{
	while(true)
	{
		int random = ((double)rand())/(double)RAND_MAX*(3.5);

		int oldEndJunc = endJunc;
		int oldStartJunc = startJunc;

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

}

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

    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];
}

bool CAutomobile::IsCarNear()
{
        for(int i = 0; i < cars->size(); i++)
        {
                if(&(*cars)[i] != this)
                {
                        if(((*cars)[i].startJunc == startJunc) && ((*cars)[i].endJunc ==
endJunc) &&
                                (((*cars)[i].distance - distance) < 40) &&
(((*cars)[i].distance - distance) > 0))
                        {
                                return true;
                        }
                }
        }

        return false;
}

bool CAutomobile::IsJunction()
{
        int distanceBetweenJuncs = sqrt(pow((*juncs)[endJunc].y -
(*juncs)[startJunc].y,2)+pow((*juncs)[endJunc].x - (*juncs)[startJunc].x,2));

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

        return false;
}

bool CAutomobile::IsJunctionNear()
{
        int distanceBetweenJuncs = sqrt(pow((*juncs)[endJunc].y -
(*juncs)[startJunc].y,2)+pow((*juncs)[endJunc].x - (*juncs)[startJunc].x,2));

        if((distanceBetweenJuncs - distance) < 60)
        {
                return true;
        }

        return false;
}
```

```
int CAutomobile::GetTurnFromDirection(int d)
{
		return 0;
}

bool CAutomobile::IsGreenLight()
{
	int direction = GetDirection();

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

bool CAutomobile::IsJuncFree()
{
	int direction = GetDirection();

	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;
}
```

Приложение 4. Класс `CJunction`

Junction.h

```
#pragma once

#include "Vector"

// Состояния
#define JUNC_STATE_RED                          2
#define JUNC_STATE_YELLOW_TO_RED                1
#define JUNC_STATE_YELLOW_TO_GREEN              3
#define JUNC_STATE_GREEN                        0

// Копии названий состояний
#define JUNC_STATE_RED_NS                       2
#define JUNC_STATE_YELLOW_TO_RED_NS             1
#define JUNC_STATE_YELLOW_TO_GREEN_NS           3
#define JUNC_STATE_GREEN_NS                     0

#define JUNC_STATE_RED_EW                       0
#define JUNC_STATE_YELLOW_TO_RED_EW             3
#define JUNC_STATE_YELLOW_TO_GREEN_EW           1
#define JUNC_STATE_GREEN_EW                     2

// События
#define EVENT_CLOCK                             0

// Типы перекрестков
#define JUNC_TYPE_NO_TL                         -1
#define JUNC_TYPE_X                             0
#define JUNC_TYPE_T_N                           1
#define JUNC_TYPE_T_W                           2
#define JUNC_TYPE_T_S                           3
#define JUNC_TYPE_T_E                           4

class CJunction
{
public:
        int y1;

        int x, y;
        int neighbours[4];

        int type;

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

        CJunction(void);
        ~CJunction(void);

        void Main(int event);
};
```

Junction.cpp

```
#include "StdAfx.h"
#include ".\junction.h"

CJunction::CJunction(void)
```

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

	y1 = JUNC_STATE_GREEN;
}

CJunction::~CJunction(void)
{
}

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

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

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

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

	switch( y1 )
	{
	case 0:// Зеленый
		break;

	case 1:// Желтый
		break;

	case 2:// Красный
		break;

	case 3:// Красный и желтый
		break;

	}
}
```