МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

РУКОВОДСТВО РАЗРАБОТЧИКА

студента 2 курса 211 группы
направления 02.03.02 — Фундаментальная информатика и информационные технологии
факультета КНиИТ
Слепова Ильи Алексеевича

Научный руководитель
доцент, к. ф.-м. н. _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2019

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 Техническое задание 4
2 Сводка файлов проекта 6
3 Подготовка к работе с проектом 7
4 Описание классов 10
4.1 Класс «Car» 10
4.2 Класс «Walker» 16
4.3 Класс «TrafficLight» 18
5 Методы, использующиеся приложением 23
5.1 Метод установки шрифта «setFont» 24
5.2 Метод изменения цвета кнопок «StartColor» 25
5.3 Метод проверки области появления автомобилей «CheckIfSpawnClear» 26
5.4 Метод проверки расстояние от пешехода до автомобиля «ChechDistanceFromWalkerToCar» 27
5.5 Метод моделирования «StartModel» 28
5.6 Метод запуска/перезапуска/окончания приложения «ModelRunning» 29
6 Описание цикла «while (window.isOpen())» 30
6.1 Меню настройки 31
6.2 Цикл «while (window.isOpen())» 32

ВВЕДЕНИЕ

В данном руководстве изложена документация для приложения «Моделирование движения на перекрестке», которая позволяет изучить различные режимы работы светофоров для поиска режима их оптимальной работы.

1 Техническое задание

На перекрестке двух автомобильных дорог расположены регулирующие движение светофоры. Каждая из дорог содержит несколько полос (рядов), автомобили двигаются в обоих направлениях. Светофоры обеспечивают проезд автомобилей по обеим дорогам, включая левый и правый повороты автомобилей, а также переход через эти дороги пешеходов.

Программа моделирования и визуализации движения на таком перекрестке служит для исследования характера возникающих на перекрестке автомобильных дорог заторов и их рассасывания в зависимости от плотностей потоков автомобилей и режимов работы светофоров.

Автомобили должны появляться на концах каждой из дорог случайным образом, проезжать по ним со скоростью, заданной при их появлении, притормаживая и останавливаясь при необходимости на перекрестке, и исчезая после проезда всей дороги на ее противоположном конце. У каждого автомобиля может быть своя начальная скорость, она определятся как случайная величина из некоторого диапазона (например, от 30 до 120 км/час). Случайной величиной является также интервал между появлениями автомобилей на каждой дороге – от диапазона изменения этой величины (и закона ее распределения) зависит плотность потока автомобилей. Как случайную величину, определяемую в момент появления автомобиля на дороге, следует моделировать и направление его проезда через перекресток (прямо / налево / направо).

Автомобили должны перестраиваться из одного ряда в другой и пересекать перекресток в соответствии с правилами дорожного движения. В частности, в левый ряд перед светофором становятся автомобили, которым необходим поворот налево. Кроме правил смены полосы, в программе должны быть зафиксированы законы торможения и ускорения автомобилей на перекрестке, которые в общем случае зависят от допустимого сближения между автомобилями, величин их скорости и др. Возможность аварий (например, из-за нарушений правил дорожного движения) в модели можно не учитывать.

Цель проводимого моделирования – изучение различных режимов работы светофоров для поиска режима их оптимальной работы. Следует рассмотреть два типа режимов работы: статический, когда интервалы свечения каждого цвета (желтый, зеленый, красный) зафиксированы заранее, и динамический, при котором интервалы свечения изменяются в соответствии с количеством

автомобилей (и пешеходов), ожидающих проезда (прохода) через дорогу.

В изменяемые параметры моделирования движения следует включить: тип режима работы светофора, интервалы свечения каждого цвета (для статического режима), дистанцию видимости светофора, диапазон возможных скоростей автомобилей, интервалы случайного появления автомобилей на каждой из дорог.

Визуальная картина движения на перекрестке дорог должна содержать изображения дорог, светофоров, движущихся машин. Полезно отобразить тем или иным образом (например, разными цветами) возможные направления движения автомобиля через перекресток (прямо/налево/направо). Желательно также предусмотреть вывод некоторых подсчитанных в ходе моделирования величин, к примеру, среднее время остановки автомобилей на перекрестке.

2 Сводка файлов проекта

Таблица 1 – Сводка всех файлов входящих в состав проекта Modeling the intersection

Название файла	Описание
Modeling the intersection	Файл, организующий проект и его элементы в решение
Cars.h	Заголовочный файл, содержащий структуру класса «Car»
Cars.cpp	Файл с реализацией основных методов класса «Car»
Walker.h	Заголовочный файл, содержащий структуру класса «Walker»
Walker.cpp	Файл с реализацией основных методов класса «Walker»
TrafficLights.h	Заголовочный файл, содержащий структуру класса «TrafficLight»
TrafficLights.cpp	Файл с реализацией основных методов класса «TrafficLight»
error.h	Заголовочный файл, содержащий структуру класса «Exception»
Main.cpp	Исходный файл приложения. Содержит код моделирования и отображения графики
Furrore.ttf	Шрифт «Furrore»
Campus.ttf	Шрифт «Campus»
bg_final_menu.jpeg	Изображение, используемое на заднем фоне в меню статистики
bg_final_menu.jpeg	Изображение, используемое на заднем фоне в меню настроек
CarDownToUpBlue.png	Изображение с синей машиной, направленной вверх
CarLeftToRightBlue.png	Изображение с синей машиной, направленной вправо
CarRightToLeftBlue.png	Изображение с синей машиной, направленной влево
CarUpToDownBlue.png	Изображение с синей машиной, направленной вниз
CarDownToUpGreen.png	Изображение с зеленой машиной, направленной вверх
CarLeftToRightGreen.png	Изображение с зеленой машиной, направленной вправо
CarRightToLeftGreen.png	Изображение с зеленой машиной, направленной влево
CarUpToDownGreen.png	Изображение с зеленой машиной, направленной вниз
CarDownToUpRed.png	Изображение с красной машиной, направленной вверх
CarLeftToRightRed.png	Изображение с красной машиной, направленной вправо
CarRightToLeftRed.png	Изображение с красной машиной, направленной влево
CarUpToDownRed.png	Изображение с красной машиной, направленной вниз
House.png	Изображение с домом
model_stat.jpeg	Изображение с задним фоном для отображения статистики во время моделирования
TrafficLight1.png	Изображение со светофором, направленным влево
TrafficLight2.png	Изображение со светофором, направленным вправо
Walker.png	Изображение с пешеходом, направленным влево
Walker2.png	Изображение с пешеходом, направленным вправо

3 Подготовка к работе с проектом

Перед началом работы с проектом необходимо подключить библиотеку SFML. Для начала нужно скачать библиотеку с официального сайта: сайт библиотеки. В данном проекте использована версия для Visual Studio 2017 64-bit. Стоит отметить, что разработка происходила в Visual Studio 2019, поэтому, для компиляции проекта необходимо зайти в настройки проекта «Project- ...» → «Properties», в разделе «General» выбрать в качестве значения для «Platform Toolset» строку «Visual Studio 2017 (v141)», затем нажать «Retarget Solution» в пункте меню «Project» и нажать «ОК», не меняя предлагаемый там выбор.

Для установки библиотеки SFML необходимо перейти в свойства проекта, а затем указать путь к некоторым папкам. Во всех изменениях должна быть выбрана конфигурация «Debug» и платформа «x64», а затем выполнить все эти действия, выбрав конфигурацию «Release».

Перейдите в свойства проекта «Project- ...» → «Properties», а затем во вкладку «C/C++» → «Общие» → «Дополнительные каталоги включаемых файлов» и измените абсолютный путь к папке «include» на свой(см. рисунок 1).

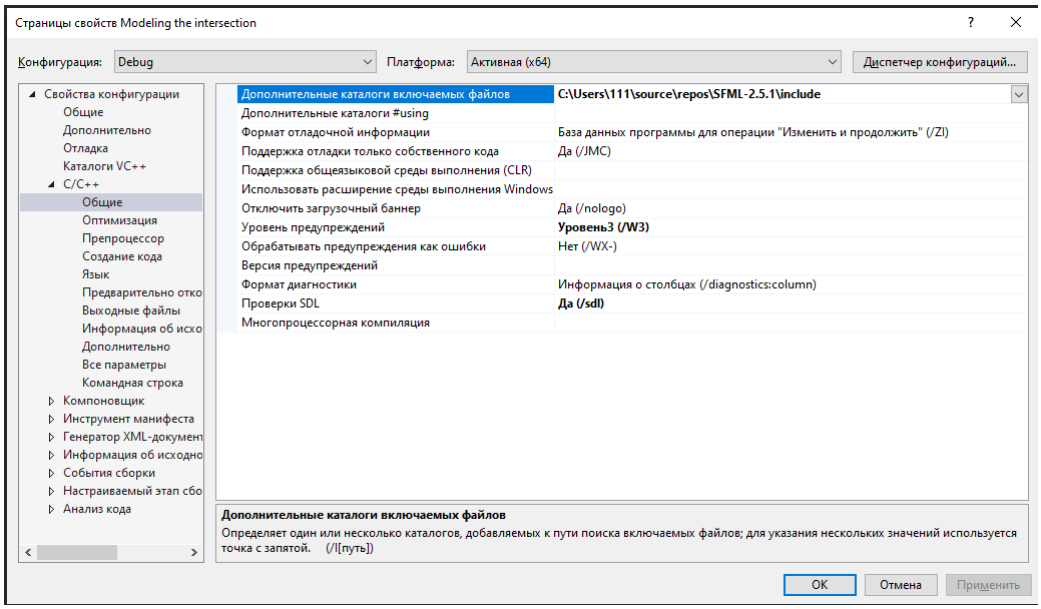

Рисунок 1 – Изменение свойств проекта — часть 1

Далее перейдите во вкладку «Компоновщик» → «Общие» → «Дополнительные каталоги библиотек» и измените абсолютный путь к папке «lib» на свой(см. рисунок 2).

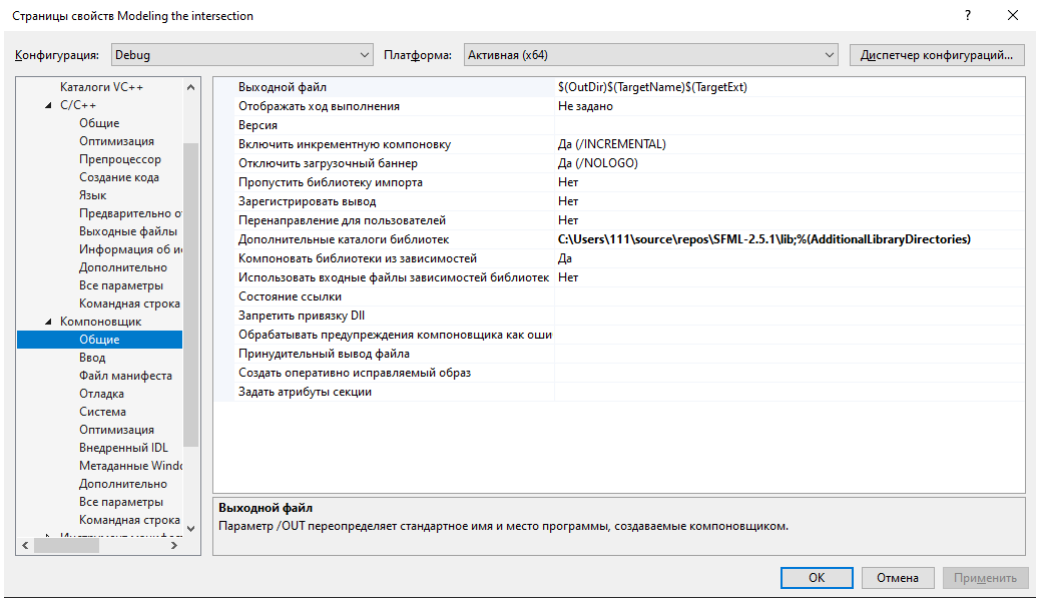

Рисунок 2 – Изменение свойств проекта — часть 2

Перейдите во вкладку «Компоновщик» → «Система» → «Подсистема» и измените подсистему на консоль(см. рисунок 3). Далее перейдите во вкладку

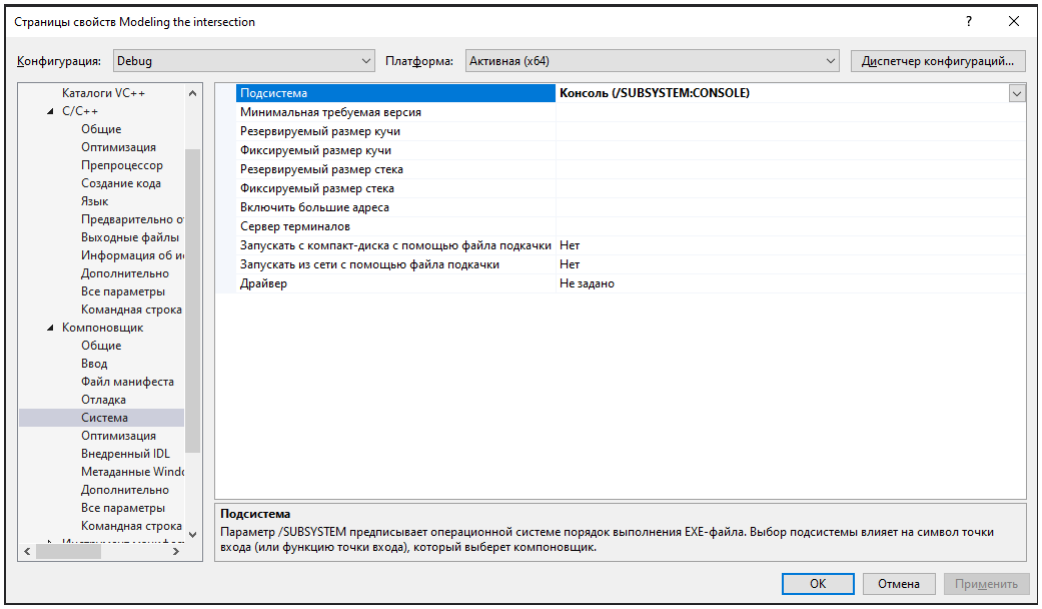

Рисунок 3 – Изменение свойств проекта — часть 3

«Компоновщик» → «Ввод» → «Дополнительные зависимости» и впишите перед всеми библиотеками строку «sfml-graphics-d.lib;sfml-window-d.lib;sfml-system-d.lib;sfml-audio-d.lib;»(см. рисунок 4).

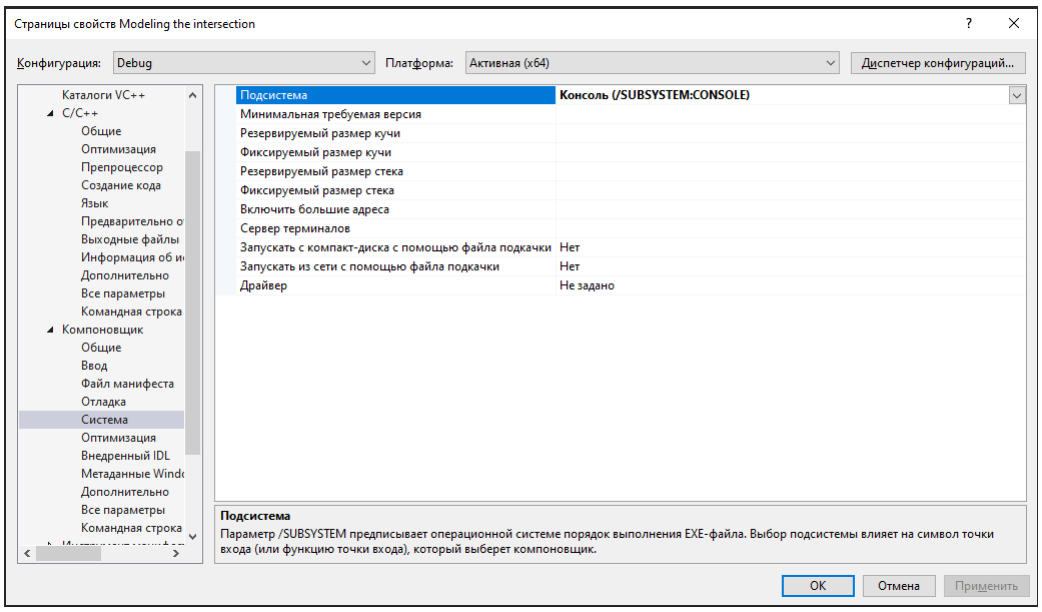

Рисунок 4 – Изменение свойств проекта — часть 4

Теперь перейдите во вкладку «C/C++» → «Препроцессор» → «Определения препроцессора» и впишите в начало строку «SFML_DYNAMIC»(см. рисунок 5).

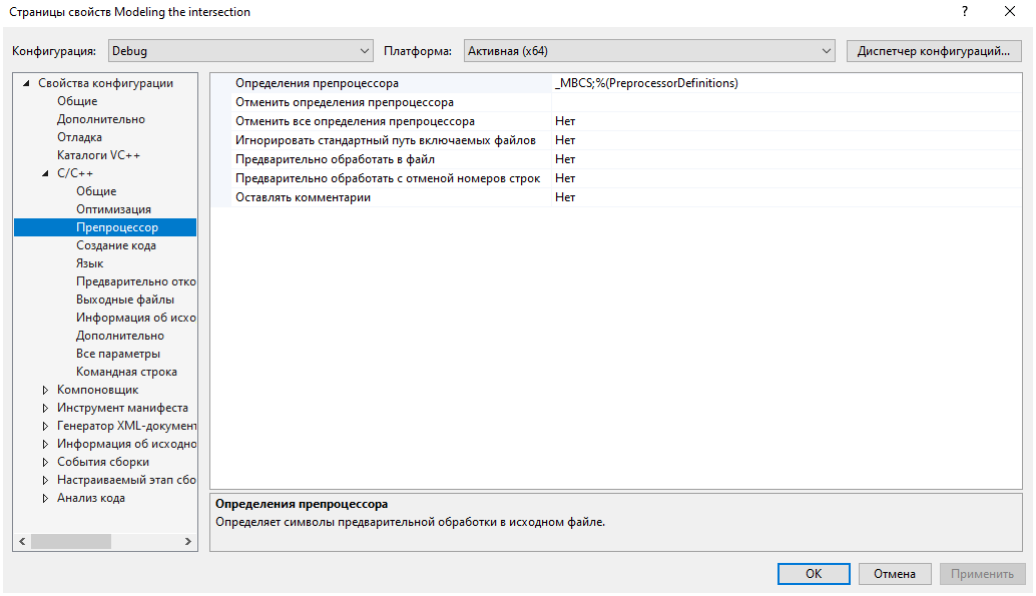

Рисунок 5 – Изменение свойств проекта — часть 5

4 Описание классов

4.1 Класс «Car»

Класс «Car» отвечает за хранение и изменение параметров автомобилей. Поля класса «Car» представлены в таблице 2.

Таблица 2 – Поля класса «Car»

Наименование поля	Модификатор доступа	Тип
Цвет автомобиля		
CarColor	private	string
Координаты автомобиля		
CurrentCarX	protected	float
CurrentCarY	protected	float
Процент пройденного пути		
DistanceToTheEndOfRoad	protected	float
Максимальная скорость автомобиля		
MaxSpeed	protected	float
Текущая скорость автомобиля		
CurrentCarSpeed	public	float
Выполнено ли перестроение		
LineChanged	public	bool
Область для проверки наличия объектов перед автомобилем		
CheckCarArea	public	RectangleShape
Текущее направление движения автомобиля		
CarDirection	public	string
Следующее направление движения автомобиля		
CarNextDirection	public	string
Хранение текстуры автомобиля		
CarImg	public	Texture
Отображение текстуры автомобиля		
CarTexture	public	Sprite

Код конструктора класса «Car» имеет следующий вид:

```
3  Car::Car(float ccx, float ccy, float s, string d, string wtt, string cc) {
4    CurrentCarX = ccx;
5    CurrentCarY = ccy;
6    MaxSpeed = s;
7    CurrentCarSpeed = MaxSpeed;
8    CarDirection = d;
9    CarNextDirection = d;
10   CarColor = cc;
11   CheckCarArea->setFillColor(Color::Green);
12   CheckCarArea->setScale(1, 6);
13   CarImg.loadFromFile(wtt);
14   CarTexture.setTexture(CarImg);
```

```
15   CarTexture.setPosition(CurrentCarX, CurrentCarY);
16   CarTexture.setScale(0.3, 0.3);
17   LineChanged = false;
18 }
```

Класс «Car» имеет всего один геттер «GetCarColor» типа «String», который возвращает цвет автомобиля:

```
35 string Car::GetCarColor() {
36   return CarColor;
37 }
```

Также данный класс имеет всего один сеттер «SetTexture» типа «Void», который устанавливает текстуру автомобиля:

```
127 void Car::SetTexture(string WayToCarTexture) {
128   CarImg.loadFromFile(WayToCarTexture);
129   CarTexture.setTexture(CarImg);
130   CarTexture.setPosition(CurrentCarX, CurrentCarY);
131   CarTexture.setScale(0.3, 0.3);
132 }
```

Ключевым методом класса «Car» является «Moving» типа «Void». Данный метод выполняет перемещение текстуры автомобиля на экране. Код этого метода:

```
40 void Car::Moving(float XSpeed, float YSpeed) {
41   CurrentCarX += XSpeed / 10;
42   CurrentCarY += YSpeed / 10;
43   CarTexture.setPosition(CurrentCarX, CurrentCarY);
44   if (CarDirection == "RightToLeft") {
45     CheckCarArea->setRotation(125);
46     CheckCarArea->setPosition(CarTexture.getPosition() + Vector2f(20, 20));
47     DistanceToTheEndOfRoad = abs(15 - CarTexture.getPosition().x) * 100 / 1100;
       ↪ //100%/1100=x%/100;
48   }
49   else if (CarDirection == "LeftToRight") {
50     CheckCarArea->setRotation(-55);
51     CheckCarArea->setPosition(CarTexture.getPosition() + Vector2f(90, 50));
52     DistanceToTheEndOfRoad = abs(1010 - CarTexture.getPosition().x) * 100 / 1100;
53   }
54   else if (CarDirection == "DownToUp") {
55     CheckCarArea->setRotation(-125);
56     CheckCarArea->setPosition(CarTexture.getPosition() + Vector2f(80, 40));
```

```
57      DistanceToTheEndOfRoad = abs(0 - CarTexture.getPosition().y) * 100 / 720;
58    }
59    else {
60      CheckCarArea->setRotation(52);
61      CheckCarArea->setPosition(CarTexture.getPosition() + Vector2f(40, 60));
62      DistanceToTheEndOfRoad = abs(720 - CarTexture.getPosition().y) * 100 / 720;
63    }
64  }
```

Торможение автомобиля выполняется с помощью метода «Braking» типа «Void». Его код представлен ниже:

```
21  void Car::Braking() {
22    if (CurrentCarSpeed > 0)
23      CurrentCarSpeed -= MaxSpeed * 0.01;
24    else
25      CurrentCarSpeed = 0;
26  }
```

Ускорение автомобиля выполняется с помощью метода «Acceleration» типа «Void». Код этого метода:

```
29  void Car::Acceleration() {
30    if (CurrentCarSpeed < MaxSpeed)
31      CurrentCarSpeed += 0.0005;
32  }
```

Метод «ChangeWay» имеет тип «Void» и используется для выбора следующего направления автомобиля. Реализация данного метода:

```
67  void Car::ChangeWay() {
68    if (CarDirection == "LeftToRight") {
69      int Rotate = rand() % 3 + 1;
70      switch (Rotate) {
71      case 1: //вверх
72        CarNextDirection = "DownToUp";
73        break;
74      case 2: //Вниз
75        CarNextDirection = "UpToDown";
76        break;
77      case 3:
78        CarNextDirection = CarDirection;
79        break;
80      }
81    }
```

```
82    else if (CarDirection == "RightToLeft") {
83      int Rotate = rand() % 3 + 1;
84      switch (Rotate) {
85      case 1: //вверх
86        CarNextDirection = "DownToUp";
87        break;
88      case 2: //Вниз
89        CarNextDirection = "UpToDown";
90        break;
91      case 3:
92        CarNextDirection = CarDirection;
93        break;
94      }
95    }
96    else if (CarDirection == "UpToDown") {
97      int Rotate = rand() % 3 + 1;
98      switch (Rotate) {
99      case 1: //влево
100       CarNextDirection = "LeftToRight";
101       break;
102     case 2: //Вниз
103       CarNextDirection = "RightToLeft";
104       break;
105     case 3:
106       CarNextDirection = CarDirection;
107       break;
108     }
109   }
110   else if (CarDirection == "DownToUp") {
111     int Rotate = rand() % 3 + 1;
112     switch (Rotate) {
113     case 1: //влево
114       CarNextDirection = "LeftToRight";
115       break;
116     case 2: //Вниз
117       CarNextDirection = "RightToLeft";
118       break;
119     case 3:
120       CarNextDirection = CarDirection;
121       break;
122     }
123   }
124 }
```

Метод «IsOppositeDirections» имеет тип «bool» и определяет, противоположны ли направления данного автомобиля и другого транспортного средства, или же данного автомобиля и пешехода. Код данного метода:

```
135 bool Car::IsOppositeDirections(string Dir) {
136   if ((Dir == "DownToUp" && CarDirection == "UpToDown") || (CarDirection == "DownToUp" &&
      ↪  Dir == "UpToDown") || (Dir == "LeftToRight" && CarDirection == "RightToLeft") ||
      ↪  (CarDirection == "LeftToRight" && Dir == "RightToLeft"))
137     return true;
138   return false;
139 }
```

Метод «CheckAheadCarSpeed» имеет тип «bool» и проверяет зону перед автомобилем на наличие других автомобилей, чтобы определить, требуется ли остановка. Его реализация:

```
142 bool Car::CheckAheadCarSpeed(vector <Car*> Cars, string TrafficLightSignal) {
143   for (int i = 0; i < Cars.size(); i++) {
144     if (CheckCarArea->getGlobalBounds().intersects(Cars[i]->CarTexture.getGlobalBounds())
      ↪  &&
      ↪  Cars[i]->CheckCarArea->getGlobalBounds().intersects(CarTexture.getGlobalBounds())
      ↪  && Cars[i]->CarTexture.getGlobalBounds() != CarTexture.getGlobalBounds()) { //Если
      ↪  машины пересекаются друг с другом
145       return false;
146     }
147     //Если  зона проверки машины пересекает другую и они находятся на одной дороге в одном
      ↪  направлении
148     else if
      ↪  (CheckCarArea->getGlobalBounds().intersects(Cars[i]->CarTexture.getGlobalBounds())
      ↪  && Cars[i]->CarTexture.getGlobalBounds() != CarTexture.getGlobalBounds() &&
      ↪  Cars[i]->CarDirection == CarDirection && Cars[i]->LineChanged == LineChanged) {
149       return true;
150     }
151     //Если зона проверки машины пересекает другую и этой машине осталось ехать до конца
      ↪  дороги меньше
152     else if
      ↪  (CheckCarArea->getGlobalBounds().intersects(Cars[i]->CarTexture.getGlobalBounds())
      ↪  && Cars[i]->CarTexture.getGlobalBounds() != CarTexture.getGlobalBounds() &&
      ↪  DistanceToTheEndOfRoad > Cars[i]->DistanceToTheEndOfRoad) {
153       if (IsOppositeDirections(Cars[i]->CarDirection)) //Если направления противоположны
154         return false;
155       return true;
156     }
157   }
158   return false;
159 }
```

Метод «CheckWalker» имеет тип «Void» и действует аналогично ранее описанному методу «CheckAheadCarSpeed», но проверяет зону перед автомобилем на наличие пешеходов. Код данного метода:

```
162 void Car::CheckWalker(vector <Walker*> Walkers) {
163   for (int i = 0; i < Walkers.size(); i++)
164     if
      ↪  (CheckCarArea->getGlobalBounds().intersects(Walkers[i]->WalkerTexture.getGlobalBounds())
      ↪  && CarDirection != Walkers[i]->WalkerDirection &&
      ↪  !IsOppositeDirections(Walkers[i]->WalkerDirection))
165       Braking();
166 }
```

4.2 Класс «Walker»

Класс «Walker» отвечает хранение и изменение параметров пешеходов. Поля класса «Walker» представлены в таблице 3.

Таблица 3 – Поля класса «Walker»

Наименование поля	Модификатор доступа	Тип
Хранение текстуры пешехода		
WalkerImg	private	Texture
Координаты пешехода		
CurrentWalkerX	protected	float
CurrentWalkerY	protected	float
Отображение текстуры пешехода		
WalkerTexture	public	Sprite
Максимальная скорость пешехода		
Speed	public	float
Текущая скорость пешехода		
CurrentWalkerSpeed	public	float
Направление движения пешехода		
WalkerDirection	public	string
Область для проверки наличия объектов перед пешеходом		
CheckWalkerArea	public	RectangleShape

Код конструктора класса «Walker» имеет следующий вид:

```
3  Walker::Walker(float s, float cwx, float cwy, string d, string WayToWalkerTexture) {
4    CurrentWalkerX = cwx;
5    CurrentWalkerY = cwy;
6    Speed = s;
7    CurrentWalkerSpeed = Speed;
8    WalkerDirection = d;
9    WalkerImg.loadFromFile(WayToWalkerTexture);
10   WalkerTexture.setTexture(WalkerImg);
11   WalkerTexture.setPosition(CurrentWalkerX, CurrentWalkerY);
12   WalkerTexture.setScale(0.3, 0.3);
13   CheckWalkerArea->setFillColor(Color::Green);
14 }
```

Ключевым методом класса «Walker» является «Moving» типа «Void». Данный метод выполняет перемещение текстуры пешехода на экране. Код этого метода:

```
17 void Walker::Moving(float XSpeed, float YSpeed) {
18   CurrentWalkerX += XSpeed / 10;
19   CurrentWalkerY += YSpeed / 10;
```

```
20    WalkerTexture.setPosition(CurrentWalkerX, CurrentWalkerY);
21
22    //Передвижение зоны проверки пешехода
23    if (WalkerDirection == "DownToUp") {
24      CheckWalkerArea->setRotation(0);
25      CheckWalkerArea->setPosition(WalkerTexture.getPosition() + Vector2f(20, 0));
26    }
27    else if (WalkerDirection == "UpToDown") {
28      CheckWalkerArea->setRotation(0);
29      CheckWalkerArea->setPosition(WalkerTexture.getPosition() + Vector2f(-20, 10));
30    }
31    else if (WalkerDirection == "LeftToRight") {
32      CheckWalkerArea->setRotation(0);
33      CheckWalkerArea->setPosition(WalkerTexture.getPosition() + Vector2f(20, 10));
34    }
35    else {
36      CheckWalkerArea->setRotation(0);
37      CheckWalkerArea->setPosition(WalkerTexture.getPosition() + Vector2f(-10, 0));
38    }
39  }
```

Метод «CheckDistanceToAnotherWalker» имеет тип «bool» и проверяет область перед пешеходом на наличие пешехода, чтобы определить, требуется ли остановка. Его реализация:

```
42  bool Walker::CheckDistanceToAnotherWalker(vector <Walker*> Walkers, string
    ↪  TrafficLightSignal) {
43    for (int i = 0; i < Walkers.size(); i++) {
44      if
        ↪  (CheckWalkerArea->getGlobalBounds().intersects(Walkers[i]->WalkerTexture.getGlobalBounds())
        ↪  && Walkers[i]->WalkerTexture.getGlobalBounds() != WalkerTexture.getGlobalBounds()
        ↪  && WalkerDirection == Walkers[i]->WalkerDirection && TrafficLightSignal == "Red")
        ↪  { //Если впереди пешеход и сигнал светофора - красный
45        CurrentWalkerSpeed = 0;
46        return true;
47      }
48    }
49    CurrentWalkerSpeed = Speed;
50    return false;
51  }
```

4.3 Класс «TrafficLight»

Класс «TrafficLight» отвечает за хранение и изменение переменных светофоров.

Поля класса «TrafficLight» представлены в таблице 4.

Таблица 4 – Поля класса «TrafficLight»

Наименование поля	Модификатор доступа	Тип
Хранение текстуры светофора		
TrafficLightImage	private	Texture
Множество ожидающих пешеходов		
WalkersCount	protected	set <Walker*>
Множество ожидающих автомобилей		
CarsCount	protected	set <Car*>
Режим работы светофора		
TrafficlightMode	protected	string
Текущий сигнал светофора		
CurrentSignal	protected	string
Дальность видимости светофора		
ViewDistance	protected	float
Задержка смены сигнала светофора при работе в динамическом режиме		
DynamicModeSwitchDelay	protected	int
Задержка смены сигнала светофора при работе в статическом режиме		
SwitchDelay	protected	float
Отображение текстуры светофора		
TrafficLightTexture	public	Sprite
Отображение зеленого и красного сигналов светофора		
Signal	public	CircleShape
Отображение желтого сигнала светофора		
YellowSignal	public	CircleShape

Код конструктора класса «TrafficLight» имеет следующий вид:

```
3  TrafficLight::TrafficLight(string tm, float sd, float vd, string cs, string wts, float
   ↪  xpos, float ypos) {
4    TrafficlightMode = tm;
5    SwitchDelay = sd;
6    DynamicModeSwitchDelay = sd;
7    ViewDistance = vd;
8    CurrentSignal = cs;
9    TrafficLightImage.loadFromFile(wts);
10   TrafficLightTexture.setTexture(TrafficLightImage);
11   TrafficLightTexture.setPosition(xpos, ypos);
12   TrafficLightTexture.setScale(0.7, 1);
13   Signal->setFillColor(Color::Transparent);
14   YellowSignal->setFillColor(Color::Transparent);
15 }
```

Данный класс имеет три гетера. Первый: «GetSwitchDelay», который имеет тип «float» и возвращает задержку смены сигнала светофора. Его реализация:

```
75 float TrafficLight::GetSwitchDelay() {
76   return SwitchDelay;
77 }
```

Вторым гетером этого класса является метод «GetTrafficlightMode» типа «string». Данный метод возвращает режим работы светофора. Код этого метода:

```
80 string TrafficLight::GetTrafficlightMode() {
81   return TrafficlightMode;
82 }
```

Последним гетером класса «TrafficLight» является метод «GetCurrentSignal», имеющий тип «string» и возвращающий текущий сигнал светофора. Реализация данного метода:

```
85 string TrafficLight::GetCurrentSignal() {
86   return CurrentSignal;
87 }
```

Первым из двух ключевых методов класса «TrafficLight» является «SwitchSignalStatic» типа «bool». Данный метод изменяет сигнал светофора при работе в статическом режиме. Код этого метода:

```
18 bool TrafficLight::SwitchSignalStatic(string Current, float SwitchTimerSeconds) {
19   bool Swithced = false;
20   YellowSignal->setFillColor(Color::Transparent);
21   if (SwitchTimerSeconds >= SwitchDelay) {
22     if (CurrentSignal == "Green") {
23       Signal->setFillColor(Color::Red);
24       Signal->setPosition(Signal->getPosition() - Vector2f(0, 47));
25       CurrentSignal = "Red";
26     }
27     else if (CurrentSignal == "Red") {
28       Signal->setFillColor(Color::Green);
29       Signal->setPosition(Signal->getPosition() + Vector2f(0, 47));
30       CurrentSignal = "Green";
31     }
32     Swithced = true;
33   }
```

```
34   return Swithced;
35 }
```

Вторым ключевым методом класса «TrafficLight» является «SwitchSignalDynamic» типа «bool». Его реализация:

```
38 bool TrafficLight::SwitchSignalDynamic(string Current, float SwitchTimerSeconds, vector
   ↪  <TrafficLight*> TrafficLights) {
39   bool Swithced = false;
40   bool IsSet = true;
41   YellowSignal->setFillColor(Color::Transparent);
42   if (SwitchTimerSeconds > DynamicModeSwitchDelay) {
43     if (CurrentSignal == "Green") {
44       Signal->setFillColor(Color::Red);
45       Signal->setPosition(Signal->getPosition() - Vector2f(0, 47));
46       CurrentSignal = "Red";
47     }
48     else if (CurrentSignal == "Red") {
49       Signal->setFillColor(Color::Green);
50       Signal->setPosition(Signal->getPosition() + Vector2f(0, 47));
51       CurrentSignal = "Green";
52     }
53     Swithced = true;
54     IsSet = false;
55
56
57   }
58   if (SwitchTimerSeconds >= DynamicModeSwitchDelay - 2) {
59     if (!IsSet) {
60       TrafficLights[0]->DynamicModeSwitchDelaySet(TrafficLights);
61       IsSet = true;
62     }
63     YellowSignal->setFillColor(Color::Yellow);
64     if (CurrentSignal == "Red")
65       YellowSignal->setPosition(Signal->getPosition() + Vector2f(0, 23));
66     if (CurrentSignal == "Green")
67       YellowSignal->setPosition(Signal->getPosition() - Vector2f(0, 23));
68     DynamicModeSwitchDelay = TrafficLights[0]->DynamicModeSwitchDelay;
69   }
70
71   return Swithced;
72 }
```

Вторым ключевым методом класса «TrafficLight» является «SwitchSignalDynamic» типа «bool». Его реализация:

```
38 bool TrafficLight::SwitchSignalDynamic(string Current, float SwitchTimerSeconds, vector
   ↪  <TrafficLight*> TrafficLights) {
39   bool Swithced = false;
40   bool IsSet = true;
41   YellowSignal->setFillColor(Color::Transparent);
42   if (SwitchTimerSeconds > DynamicModeSwitchDelay) {
43     if (CurrentSignal == "Green") {
44       Signal->setFillColor(Color::Red);
45       Signal->setPosition(Signal->getPosition() - Vector2f(0, 47));
46       CurrentSignal = "Red";
47     }
48     else if (CurrentSignal == "Red") {
49       Signal->setFillColor(Color::Green);
50       Signal->setPosition(Signal->getPosition() + Vector2f(0, 47));
51       CurrentSignal = "Green";
52     }
53     Swithced = true;
54     IsSet = false;
55
56
57   }
58   if (SwitchTimerSeconds >= DynamicModeSwitchDelay - 2) {
59     if (!IsSet) {
60       TrafficLights[0]->DynamicModeSwitchDelaySet(TrafficLights);
61       IsSet = true;
62     }
63     YellowSignal->setFillColor(Color::Yellow);
64     if (CurrentSignal == "Red")
65       YellowSignal->setPosition(Signal->getPosition() + Vector2f(0, 23));
66     if (CurrentSignal == "Green")
67       YellowSignal->setPosition(Signal->getPosition() - Vector2f(0, 23));
68     DynamicModeSwitchDelay = TrafficLights[0]->DynamicModeSwitchDelay;
69   }
70
71   return Swithced;
72 }
```

Метод «WalkersCounter» имеет тип «void» и подсчитывает количество ожидающих пешеходов. Реализация данного метода:

```
90 void TrafficLight::WalkersCounter(Walker* Walk) {
91   if (CurrentSignal == "Red") {
92     WalkersCount.insert(Walk);
93   }
94 }
```

Метод «CarsCounter» имеет тип «void» и отвечает за подсчет ожидающих автомобилей. Код этого метода:

```
97  void TrafficLight::CarsCounter(Car* Car) {
98    if (CurrentSignal == "Red") {
99      CarsCount.insert(Car);
100   }
101 }
```

Последним стал метод «DynamicModeSwitchDelaySet» типа «void». Данный метод отвечает за вычисление задержки смены сигнала при динамическом режиме работы светофора. Его реализация:

```
104 void TrafficLight::DynamicModeSwitchDelaySet(vector <TrafficLight*> TrafficLights) {
105   if (TrafficLights[0]->CarsCount.size() + TrafficLights[0]->WalkersCount.size() >
      ↪  TrafficLights[1]->CarsCount.size() + TrafficLights[1]->WalkersCount.size()) //Если
      ↪  ожидающих на горизонтальной дороге больше
106     DynamicModeSwitchDelay = TrafficLights[0]->CarsCount.size() +
        ↪  TrafficLights[0]->WalkersCount.size() + 5;
107   else //Если ожидающих на вертикальной дороге больше
108     DynamicModeSwitchDelay = TrafficLights[1]->CarsCount.size() +
        ↪  TrafficLights[1]->WalkersCount.size() + 5;
109
110   //Очистка множеств
111   WalkersCount.clear();
112   CarsCount.clear();
113 }
```

5 Методы, использующиеся приложением

Приложение «Моделирование движения на перекрестке», кроме ранее описанных методов классов, использует 6 методов. В данном разделе будут описаны они все.

5.1 Метод установки шрифта «setFont»

Данный метод имеет тип «Font» и устанавливает шрифт для отображения текста на экране. Его реализация:

```
Font setFont(string fname) { //Установка шрифта
  Font font;
  try {
    bool isOk = font.loadFromFile(fname);
    if (!isOk)
      throw(fname);
  }
  catch (...) {
    cerr << "Файл, содержащий шрифт " << fname << ", не найден";
  }
  return font;
}
```

5.2 Метод изменения цвета кнопок «StartColor»

Данный метод типа «void» устанавливает изначальный цвет кнопки после его изменения при нажатии. Код данного метода:

```
22 void StartColor(list <RectangleShape*> Buttons) { //Изменение цвета кнопок на начальный
   ↪ после нажатия
23   for (auto i : Buttons)
24     i->setFillColor(Color(50, 50, 50, 255));
25 }
```

5.3 Метод проверки области появления автомобилей «CheckIfSpawnClear»

Этот метод имеет тип «bool» и проверяет области «CheckSpawnAreas» на отсутствие в них автомобилей с определенным направлением. Если автомобиль находится в данной области - другой появиться не сможет. Реализация данного метода:

```
27 bool CheckIfSpawnClear(string Direction, vector <RectangleShape*> CheckSpawnAreas, vector
   ↪ <Car*> Cars) { //Зоны для проверки появления машин
28   for (int i = 0; i < Cars.size(); i++) {
29     if (Direction == "DownToUp" && Cars[i]->CarDirection == Direction &&
       ↪ CheckSpawnAreas[0]->getGlobalBounds().intersects(Cars[i]->CarTexture.getGlobalBounds()))
30       return false;
31     else if (Direction == "RightToLeft" && Cars[i]->CarDirection == Direction &&
       ↪ CheckSpawnAreas[1]->getGlobalBounds().intersects(Cars[i]->CarTexture.getGlobalBounds()))
32       return false;
33     else if (Direction == "UpToDown" && Cars[i]->CarDirection == Direction &&
       ↪ CheckSpawnAreas[2]->getGlobalBounds().intersects(Cars[i]->CarTexture.getGlobalBounds()))
34       return false;
35     else if (Direction == "LeftToRight" && Cars[i]->CarDirection == Direction &&
       ↪ CheckSpawnAreas[3]->getGlobalBounds().intersects(Cars[i]->CarTexture.getGlobalBounds()))
36       return false;
37   }
38   return true;
39 }
```

5.4 Метод проверки расстояние от пешехода до автомобиля «ChechDistanceFromWalkerToCar»

Данный метод проверяет область «CheckWalkerArea» описанную в классе «Walker» перед пешеходом на наличие в ней автомобиля, если автомобиль есть, то пешеход не может продолжать движение. Код этого метода:

```
41 bool ChechDistanceFromWalkerToCar(Walker* Walk, vector <Car*> Cars) { //Расстояние от
   ↪ пешехода до машины
42   for (int i = 0; i < Cars.size(); i++) {
43     if
       ↪ (Walk->CheckWalkerArea->getGlobalBounds().intersects(Cars[i]->CarTexture.getGlobalBounds())
       ↪ && !Cars[i]->IsOppositeDirections(Walk->WalkerDirection) && Cars[i]->CarDirection
       ↪ != Walk->WalkerDirection) {
44       if
         ↪ (Walk->CheckWalkerArea->getGlobalBounds().intersects(Cars[i]->CarTexture.getGlobalBounds()
         ↪ &&
         ↪ Cars[i]->CheckCarArea->getGlobalBounds().intersects(Walk->WalkerTexture.getGlobalBounds())
45         break;
46       return true;
47       }
48   }
49   return false;
50 }
```

5.5 Метод моделирования «StartModel»

Данный метод имеет тип «bool» и отвечает за все моделирование. Диаграмма, иллюстрирующая логику данного метода, будет представлена позже.

5.6 Метод запуска/перезапуска/окончания приложения «ModelRunning»

Данный метод запускает и перезапускает приложение, если метод «StartModel» возвращает значение «true» типа «bool», в ином случае приложение завершает работу. Реализация данного метода:

```
1228 void ModelRunning() { //Функция запуска/перезапуска/окончания программы
1229   if (StartModel())
1230     ModelRunning();
1231 }
```

6 Описание цикла «while (window.isOpen())»

Данный цикл отвечает за вывод изображения на экран и течение моделирования, используя ранее описанные методы и классы.

6.1 Меню настройки

Меню настройки аспектов моделирования можно представить диаграммой(см. рисунок 6). Изменять параметры можно в любой последовательности, но начать моделирование до выбора режима работы нельзя.

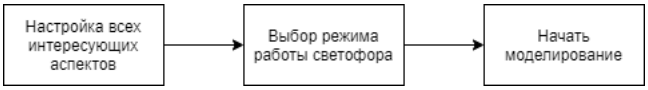

Рисунок 6 – Диаграмма меню

6.2 Цикл «while (window.isOpen())»

Диаграмма, описывающая действия в главном цикле проиллюстрирована рисунком 7).

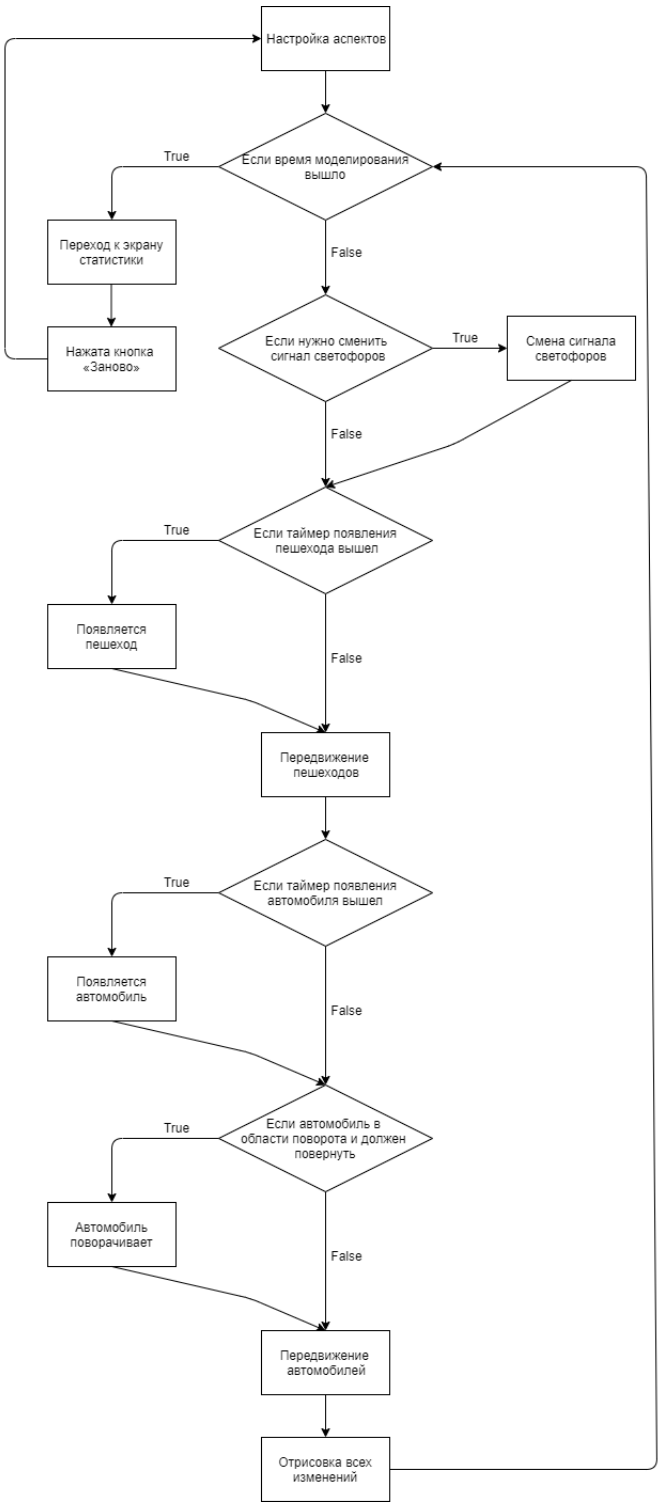

Рисунок 7 – Диаграмма, описывающая действия в главном цикле моделирования

Псевдокод вызова функций во время прохождения цикла:

```
SettingsScreen()
if ModelTime <= 0
  StatisticScreen()
  ButtonPressed->SettingsScreen()
if SignalChangeTimer <= 0
  SwitchSignal()
if SpawnWalkerTimer <= 0
  SpawnWalker()
Walker->Moving()
if SpawnCarTimer <= 0
  SpawnCar()
if Car in RotateArea && Car->CarNextDirection != Car->CarDirection
  Car->Rotate()
Car->Moving()
window.draw()
```