

Федеральное агентство связи  
Федеральное государственное бюджетное образовательное учреждение высшего  
образования  
«Сибирский государственный университет телекоммуникаций и информатики»  
(СибГУТИ)

Кафедра вычислительных систем  
Допустить к защите  
зав. кафедрой д.т.н. доцент  
\_\_\_\_\_ Курносов М.Г.

# **ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

Разработка алгоритма централизованного  
управления автомобилями для систем автоведения

Пояснительная записка

Студент: Тимофеев Д.А.

Факультет ИВТ Группа ИВ-622

Руководитель Гонцова А.В.

Новосибирск - 2020

Федеральное агентство связи  
Федеральное государственное бюджетное образовательное учреждение высшего  
образования  
«Сибирский государственный университет телекоммуникаций и информатики»  
(СибГУТИ)

**КАФЕДРА**  
**ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ**

**ЗАДАНИЕ**  
**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ**  
**БАКАЛАВРА**

СТУДЕНТУ Тимофееву Д.А.

ГРУППЫ ИВ-622

«УТВЕРЖДАЮ»

«\_\_\_\_» \_\_\_\_\_

зав. кафедрой ВС

д.т.н. доцент

\_\_\_\_\_ Курносов М.Г.

Новосибирск, 2020 г.

- 1. Тема выпускной квалификационной работы бакалавра:** «Разработка алгоритма централизованного управления автомобилями для систем автоведения» утверждена приказом СибГУТИ от «20» января 2020 г. № 4/54о-20
- 2. Срок сдачи студентом законченной работы:** 11 июня 2020 г.

**3. Исходные данные к работе**

1 Специальная литература

2 Материалы сети интернет

<b>4. Содержание пояснительной записки (перечень подлежащих разработке вопросов)</b>	<b>Сроки выполнения по разделам</b>
Постановка задачи на практику, определение конкретной индивидуальной темы, формирование плана работ. Водный инструктаж по технике безопасности (охране труда, пожарной безопасности)	03.02.20-20.02.20
Работа с библиотечными фондами, сбор и анализ материалов по теме практики. Поиск имеющегося опыта по планированию движения	20.03.20-29.02.20
Придумывание различных ситуаций и задач, при движении машин (препятствия, ландшафт и т.п.). Разработка алгоритма. Разработка архитектуры проекта в виде UML диаграмм.	20.02.20-20.03.20
Выполнение работ в соответствии с составленным планом	20.03.20-06.04.20
Реализация алгоритма и симуляции для его отладки	06.04.20-10.06.20

Дата выдачи задания: «\_\_\_» \_\_\_\_\_

Руководитель \_\_\_\_\_ Гонцова А.В.

Задание принял к исполнению «\_\_\_» \_\_\_\_\_

Студент \_\_\_\_\_ Тимофеев Д.А.

## **АННОТАЦИЯ**

Выпускная квалификационная работа Тимофеева Д.А.  
по теме «Разработка алгоритма централизованного управления автомобилями для систем автоведения»

Объём работы 43 страниц, на которых размещены 20 рисунков и 0 таблиц. При написании работы использовалось 8 источника.

Ключевые слова: вычислительная система, трансляционные обмены.

Работа выполнена на кафедре ВС СибГУТИ.

Руководитель –старший преподаватель Гонцова А.В.,

Целью бакалаврской работы было проектирование и разработка алгоритма централизованной раздачи маршрутов для систем автоведения.

Организованное движение машин по маршрутам, полученным от централизованной автоматизированной системы, сделает дорожный трафик более эффективным.

В рамках дипломного проекта был спроектирован и реализован алгоритм централизованной раздачи маршрутов для систем автоведения и симуляция для его тестирования.

Федеральное агентство связи  
Федеральное государственное бюджетное образовательное учреждение высшего  
образования  
«Сибирский государственный университет телекоммуникаций и информатики»  
(СибГУТИ)

## ОТЗЫВ

на выпускную квалификационную работу студента  
группы ИВ-622 Тимофеева Д.А.  
по теме «Разработка алгоритма централизованного управления автомобилями для  
систем автоведения»

В рамках бакалаврской работы Тимофеев Д.А. разработал алгоритм для централизованного управления движением машин, реализовал моделирование разработанного алгоритма.

Тема была предложена студентом. В ходе выполнения работы Тимофеев Д.А. показал владение необходимыми навыками, проявил самостоятельность.

Выполнение работы было затянуто по времени, не все поставленные задачи были выполнены. Разработанный алгоритм в пояснительной записке описан путано.

Оригинальность текста составляет 93,49%. Проверка осуществлялась в системе antiplagiat.ru.

Считаю, что бакалаврская работа заслуживает оценки **«хорошо»**, а Тимофеев Д.А. присвоения квалификации бакалавр по направлению 09.03.01 «Информатика и вычислительная техника», профиль «Электронно-вычислительные машины, комплексы, системы и сети».

Оценка уровней сформированности общекультурных, общепрофессиональных и профессиональных компетенций обучающегося:

Компетенции		Уровень сформированности компетенций		
		Высокий	Средний	Низкий
Общекультурные	ОК-7 - способностью к самоорганизации и самообразованию		X	
Общепрофессиональные	ОПК-4 - способностью участвовать в настройке и наладке программно-аппаратных комплексов	X		
	ОПК-5 - способностью решать стандартные задачи профессиональной деятельности на основе информационной и библиографической культуры с	X		

	применением информационно-коммуникационных технологий и с учетом основных требований информационной безопасности			
Профессиональные	ПК-1 - способностью разрабатывать модели компонентов информационных систем, включая модели баз данных и модели и интерфейсов человек – электронно-вычислительная машина	X		
	ПК-2 - способностью разрабатывать компоненты аппаратно-программных комплексов и баз данных, используя современные инструментальные средства и технологии программирования	X		
	ПК-3 - способностью обосновывать принимаемые проектные решения, осуществлять постановку и выполнять эксперименты по проверке их корректности и эффективности		X	

Работа имеет практическую ценность ☐

Работа внедрена ☐

Рекомендую работу к внедрению ☐

Рекомендую работу к опубликованию ☐

Работа выполнена с применением ЭВМ ☒

Тема предложена предприятием ☐

Тема предложена студентом ☒

Тема является фундаментальной ☐

Рекомендую студента в магистратуру ☒

Старший преподаватель  
вычислительных систем СибГУТИ

\_\_\_\_\_ Гонцова А.В.  
(Гонцова Александра Владимировна)  
« \_\_\_\_ » \_\_\_\_\_

## Содержание

1 ВВЕДЕНИЕ.....	8
1.1 Актуальность работы .....	8
1.2 Причины возникновения пробок .....	9
2 ПОСТАНОВКА ЗАДАЧИ.....	10
2.1 Задачи .....	10
2.2 Допущения и условности .....	10
2.3 Техническое задание для алгоритма .....	10
3 АЛГОРИТМ ЦКА .....	11
3.1 Алгоритм .....	11
3.2 Описание частей алгоритма .....	11
4 АРХИТЕКТУРА ПРОЕКТА .....	13
5 ПОШАГОВАЯ РАБОТА РЕАЛИЗАЦИИ АЛГОРИТМА .....	16
5.1 Подготовка карты.....	16
5.2 Построение маршрута для машины .....	16
6 ЭКСПЕРИМЕНТЫ.....	18
ПРИЛОЖЕНИЕ А .....	24
ПРИЛОЖЕНИЕ Б.....	25
ПРИЛОЖЕНИЕ В .....	26

# 1 ВВЕДЕНИЕ

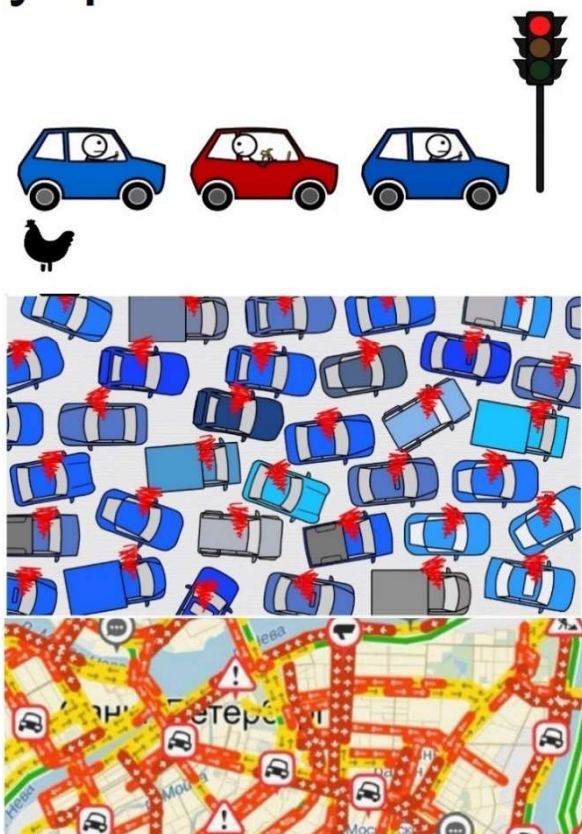
## 1.1 Актуальность работы

Централизованное управление автомобильным трафиком сделает движение на дорогах более эффективным, устранит проблему пробок, позволит быстрее перемещаться автомобилям.

В обозримом будущем машины по большей части будут управляться автопилотами.

Проблема пробок (и многие другие логистические проблемы) возникают вследствие плохо согласованного движения машин. Правила дорожного движения решают эти проблемы, но не полностью. Остается человеческий фактор (эгоистичное поведение, нарушение правил, медленная реакция). Централизованная раздача маршрутов автопилотам под управлением программы решает эти проблемы.

### Ручное управление



### Централизованные автопилоты

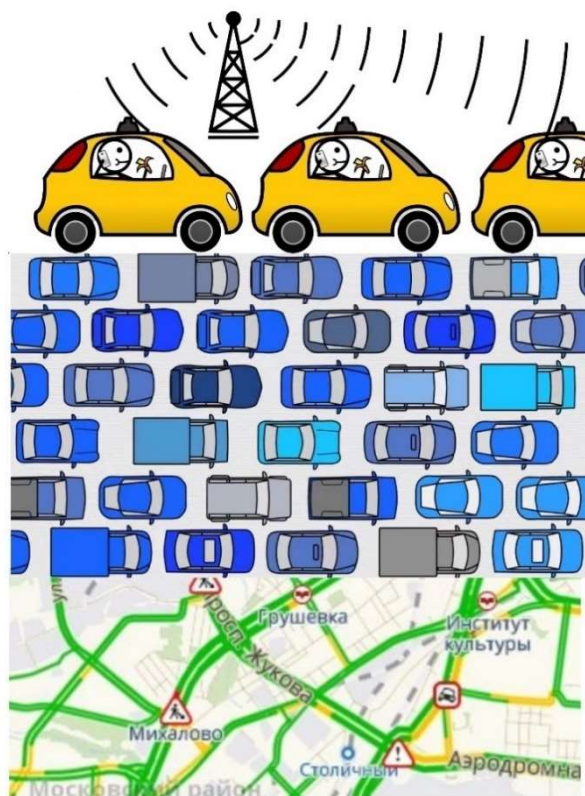


Рисунок 1.1 – Актуальность работы



## 1.2 Причины возникновения пробок

При низкой координации движения машин могут возникать фантомные перекрестки, из-за того, что кто-то притормозил, за ним следующий, а за ним следующий и т.д.

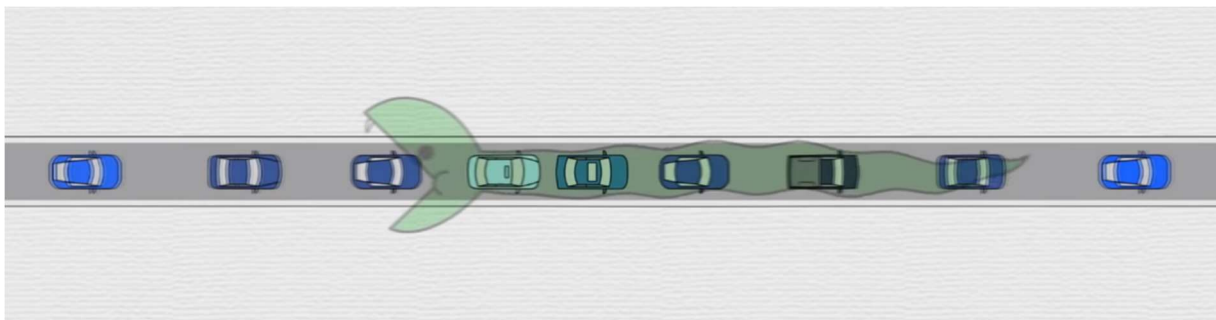


Рисунок 1.2 – Фантомный перекресток. [<https://youtu.be/xwTMmdeLRKI?t=109>]

Из-за плохой человеческой координации могут возникать пробки даже при движении на круглой замкнутой трассе без препятствий, без светофоров с несколькими машинами.

Пробки могут возникать при отсутствии препятствий, даже когда водители специально пытаются скоординироваться и не создавать пробок. Как бы водители не старались, у них не получается двигаться организованно более 5 минут. (ссылка на видеозапись проведенного эксперимента в списке литературы под названием «Видеозапись проведенного эксперимента с движением машин по кругу без препятствий»).

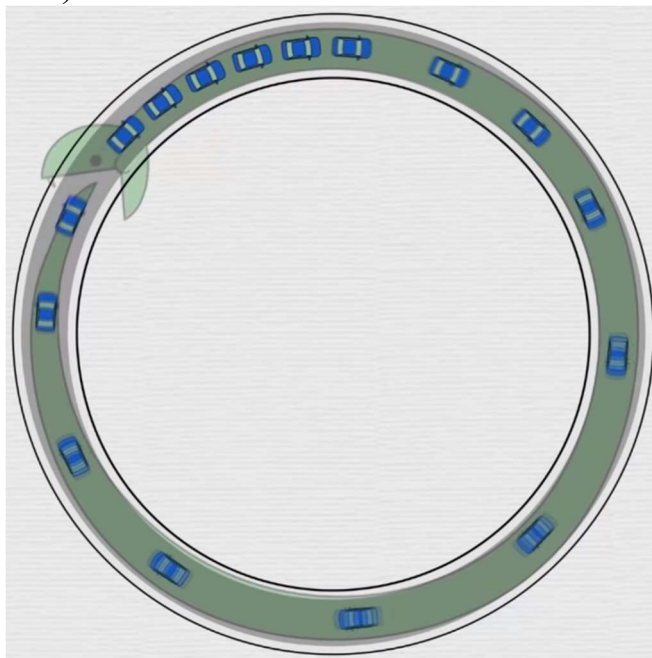


Рисунок 1.3 – Пробка на кольцевой трассе без препятствий с небольшим количеством машин. [<https://youtu.be/xwTMmdeLRKI?t=117>]

## 2 ПОСТАНОВКА ЗАДАЧИ

### 2.1 Задачи

1 Разработать централизованный логистический алгоритм (в дальнейшем называемый ЦКА (Центральный Контроль Автопилотов)).

1.1 раздать маршруты движения каждой подключенной автопилотируемой машине.

1.2 организовать движение без пробок, заторов и т.п.

2 Разработать симуляцию движения машин, для отладки и демонстрации работы алгоритма.

### 2.2 Допущения и условности

1 Машина может резко поворачивать на 90 градусов

2 Движение не обязано быть похожим на Правила Дорожного Движения.

### 2.3 Техническое задание для алгоритма

1 Пока есть возможность доехать до точки прибытия, ЦКА должен вести туда машины.

2 ЦКА должен успешно преодолевать

2.1 Статические препятствия (здания т.п.)

2.2 Динамические препятствия (неподконтрольные машины, разрушающиеся/появляющиеся внезапно здания, светофоры с кнопкой и т.п.)

3 Время отклика системы: 1 секунда (инструкции под новые условия должны появиться уже через секунду).

4 Машины к системе ЦКА могут подключаться/отключаться внезапно.

## 3 АЛГОРИТМ ЦКА

Алгоритм ЦКА (Центрального Контроля Автопилотов) раздает маршруты автопилотируемым машинам, чтобы обеспечить эффективное движение на дороге без пробок.

### 3.1 Алгоритм

Пространственно-временной A-Star с ориентацией на заранее построенный граф эталонных маршрутов.

Граф эталонных маршрутов построен при помощи алгоритма Флойда.

Под ориентацией имеется ввиду построение приоритета обхода графа, при помощи эталонных маршрутов.

### 3.2 Описание частей алгоритма

#### 3.2.1 Пространственно-временной

Это прилагательное означает, что алгоритм смотрит в точке карты не только на наличие постоянного препятствия от ландшафта, но и наличие другой машины, в данный момент времени находящейся в этом месте.

#### 3.2.2 A-Star

A-Start делает обход дерева вариантов маршрута в глубину по приоритету, основанном на весе узла. Алгоритм оценивает стоимость каждого ближайшего узла и выбирает самый короткий маршрут до узла назначения (минимальная стоимость). Потом процесс повторяется заново, относительно выбранного узла.

Стоимость узла определяется:

- 1 реальным пройденным расстоянием до узла
- 2 оценка эвристической функции, которая приблизительно определяет стоимость пути до точки прибытия

Как правило используют функцию «полет птицы», определяющую прямое расстояние между точками на плоскости. В данном проекте будет использоваться функция, ориентирующаяся на заранее построенный граф эталонных кратчайших маршрутов, построенный алгоритмом Флойда.

Более подробно об алгоритме A-Star можно прочитать по ссылке в списке литературы.

### 3.2.3 Алгоритм Флойда

Алгоритм нахождения кратчайших расстояний между всеми парами вершин во взвешенном ориентированном графе. Используется для построения эталонных маршрутов.

На каждом шаге алгоритм генерирует матрицу  $W$ . Матрица  $W$  содержит длины кратчайших путей между всеми вершинами графа. Перед работой алгоритма матрица  $W$  заполняется длинами рёбер графа (или заведомо большим  $M$ , если ребра нет).

#### Листинг 3.1 - Псевдокод алгоритма Флойда

```
for k = 1 to n
  for i = 1 to n
    for j = 1 to n
       $W[i][j] = \min(W[i][j], W[i][k] + W[k][j])$ 
```

## 4 АРХИТЕКТУРА ПРОЕКТА

В UML диаграмме указаны основные классы. Вспомогательные классы не указаны.

Все обращения к классам идут через их интерфейсы. Это не усложняет программный код, при этом оставляет возможность добавлять функционал, изменяя минимальное количество уже написанного кода.

Логика программы защищена от всех сторонних ресурсов адаптерами и интерфейсами (не имеет привязанности к конкретным отрисовщикам, системам подачи команд, фреймворкам, базам данных и т.п.).

Система сделана по принципу Model – View – Controller. А именно: цепочка вызовов функций у классов не циклична. ConsoleManager это Controller, Render – это View, все остальные классы это Model.

Для более продуктивного программирования системы были изучены и использованы паттерны ООП (адаптер, фасад, фабрика, итератор и т.п.)

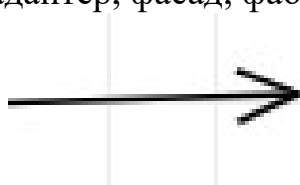


Рисунок 4.1 – Обращение к другому классу с изменением объекта.

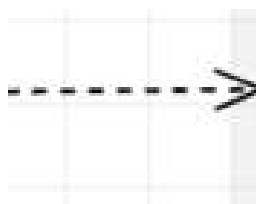


Рисунок 4.2 – Обращение к другому классу, получение данных без изменения объекта

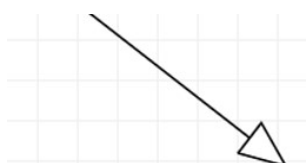


Рисунок 4.3 – Реализация интерфейса (обобщение)

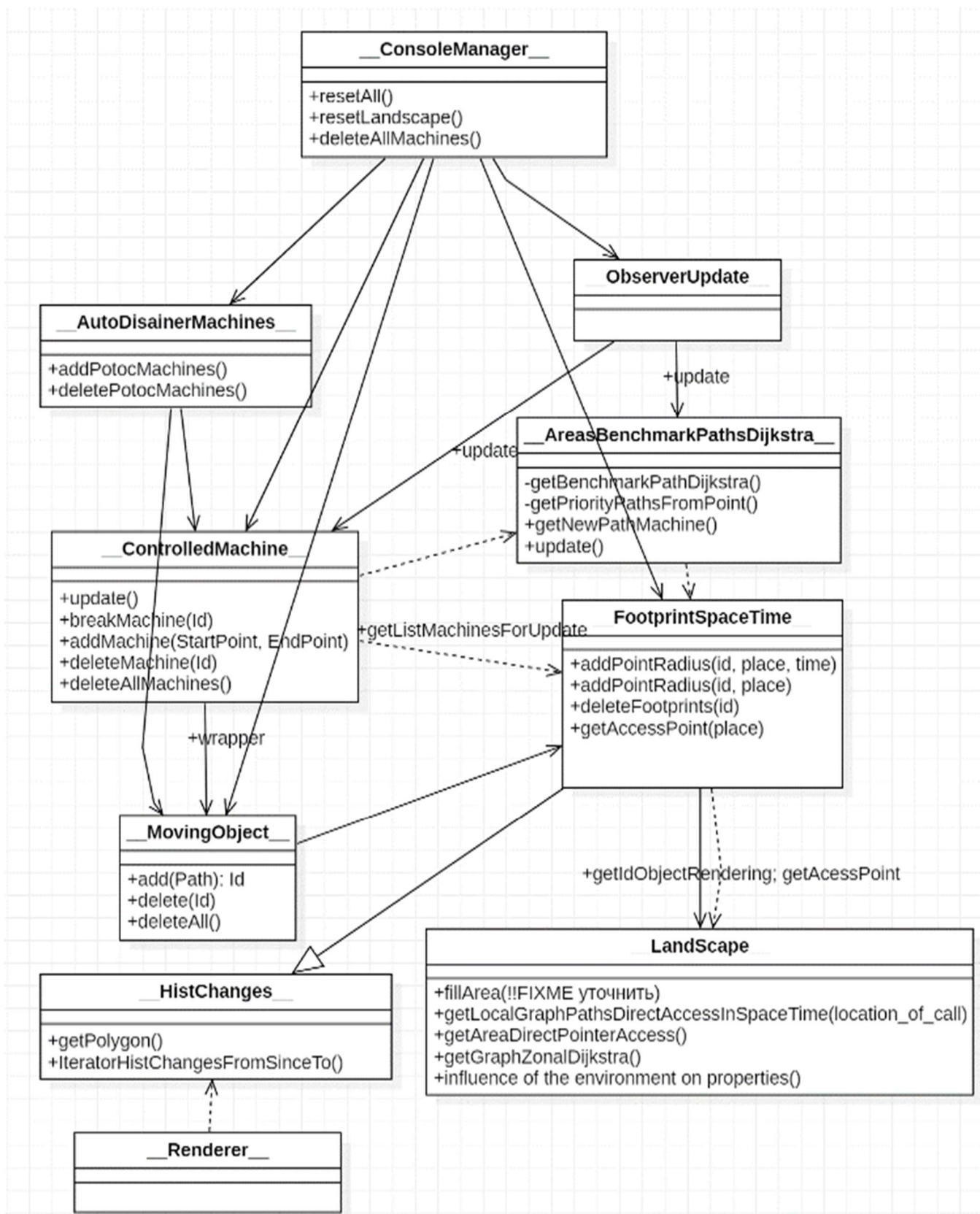


Рисунок 4.4 – UML диаграмма проекта

## 1 ConsoleManager

Используется управление системой текстовыми командами. Такой интерфейс был выбран, как максимально гибкий. ConsoleManager перерабатывает текстовые команды в вызовы функций определенных классов.

## 2 Observer

Класс Наблюдатель. ConsoleManager после некоторых изменений (например ландшафта) сообщает Наблюдателю о необходимости обновления. Наблюдатель передает оповещение об обновлении соответствующим классам в нужном порядке.

## 3 MovingObject

Класс занимается созданием объектов машин, их учетом, отвечает за оставление следов в объекте класса FootprintSpaceTime.

## 4 AutoDisainerMachines

Класс отвечает за функционирование потоков, создающих машины с заранее заданной точкой прибытия.

## 5 ControlledMachine

Класс отвечает за предоставление маршрута машине из точки А в точку В. Собственно говоря, здесь и будет работать главный логистический алгоритм (см. пункт 1).

## 6 FootprintSpaceTime

Пространственно-временные следы. Класс отвечает за

- Хранение информации о положении машины в пространстве и времени
- Выдачи информации о занятости места определенной машиной
- Решение конфликтов столкновений машин, зданий и т.п.

## 7 AreasBenchmarkPaths

Этот класс отвечает за:

- составление эталонных маршрутов из точек прямого доступа
- раздачу маршрутов автопилотируемым машинам

## 8 LandScape

Подкласс FootprintSpaceTime. Отвечает за выдачу информации о проходимости через статические объекты (неподвижные объекты).

## 9 HistChanges

Возвращает изменения за запрошенный промежуток времени.

## 10 Render

Отрисовщик определенной области карты.

## 5 ПОШАГОВАЯ РАБОТА РЕАЛИЗАЦИИ АЛГОРИТМА

Программа сделана в векторном стиле. Но при этом есть функционал, разбивающий всю карту на прямоугольную сеть узлов. Класс `NetworkNodes` возвращает ближайший узел к запрашиваемой точки. Если машина не находится на узле – то следующим шагом она доезжает до узла. Если машина находится на узле, то она едет до наиболее близкого к точке назначения узла.

Пространственно-временной след – это запись в классе `FootprintSpaceTime` в которой указывается

- ID машины (по ID запрашивается форма и размер машины)
- позиция машины в пространстве (координаты центра машины)
- Шаг равен длине машины.
- временной отрезок (время прибытия в точку и время стояния в точке),

который занимает машина (время стояния у неподвижных объектов (стена и т.п.): максимальное значение `Double`. Для движущихся объектов время стояния в одной точки зависит от скорости).

### 5.1 Подготовка карты

Карта загружается из файла. Считываются стационарные объекты (стены и т.п.). В программе есть только один класс обрабатываемых объектов – движущиеся. А стационарные объекты – это лишь частный случай движущихся объектов. Стационарные объекты – это движущиеся объекты со скоростью 0, оставляющие следы в классе `FootprintSpaceTime` с максимальным временем пребывания в точке.

Далее производится индексация карты (построение эталонных маршрутов): для каждого узла указывается наилучшее направление до каждого другого узла. Далее у класса `AreasBenchmarkPaths` будет запрашиваться эвристическая оценка расстояния от узла до точки назначения.

### 5.2 Построение маршрута для машины

Выбирается первый узел (самый близкий к начальному положению машины). Алгоритм перебирает соседние узлы, выбирая из них узел с минимальной стоимостью и не занятый в предполагаемый момент времени другой машиной (частный случай: неподвижной машиной, стеной). Информация о свободности места в данный момент времени запрашивается у класса `FootprintSpaceTime`. Стоимость узла (цифра в верхнем левом углу ячейки на рисунке 5.1) определяется реально пройденным расстоянием до узла (нижний левый угол ячейки на рисунке 5.1) и оценкой эвристической функции (нижний правый угол ячейки на рисунке 5.1), которая запрашивается у `AreasBenchmarkPaths`.

Переходы по узлам продолжаются пока не будет найден первый узел, совпавший с конечной целью или пока не закончатся все узлы.

После построения маршрута класс `MovingObject` добавляет пространственно-временные следы движения машины в класс `FootprintSpaceTime`, чтобы следующие машины при построении маршрутов могли получить точную информацию о занятости мест в разные моменты времени.



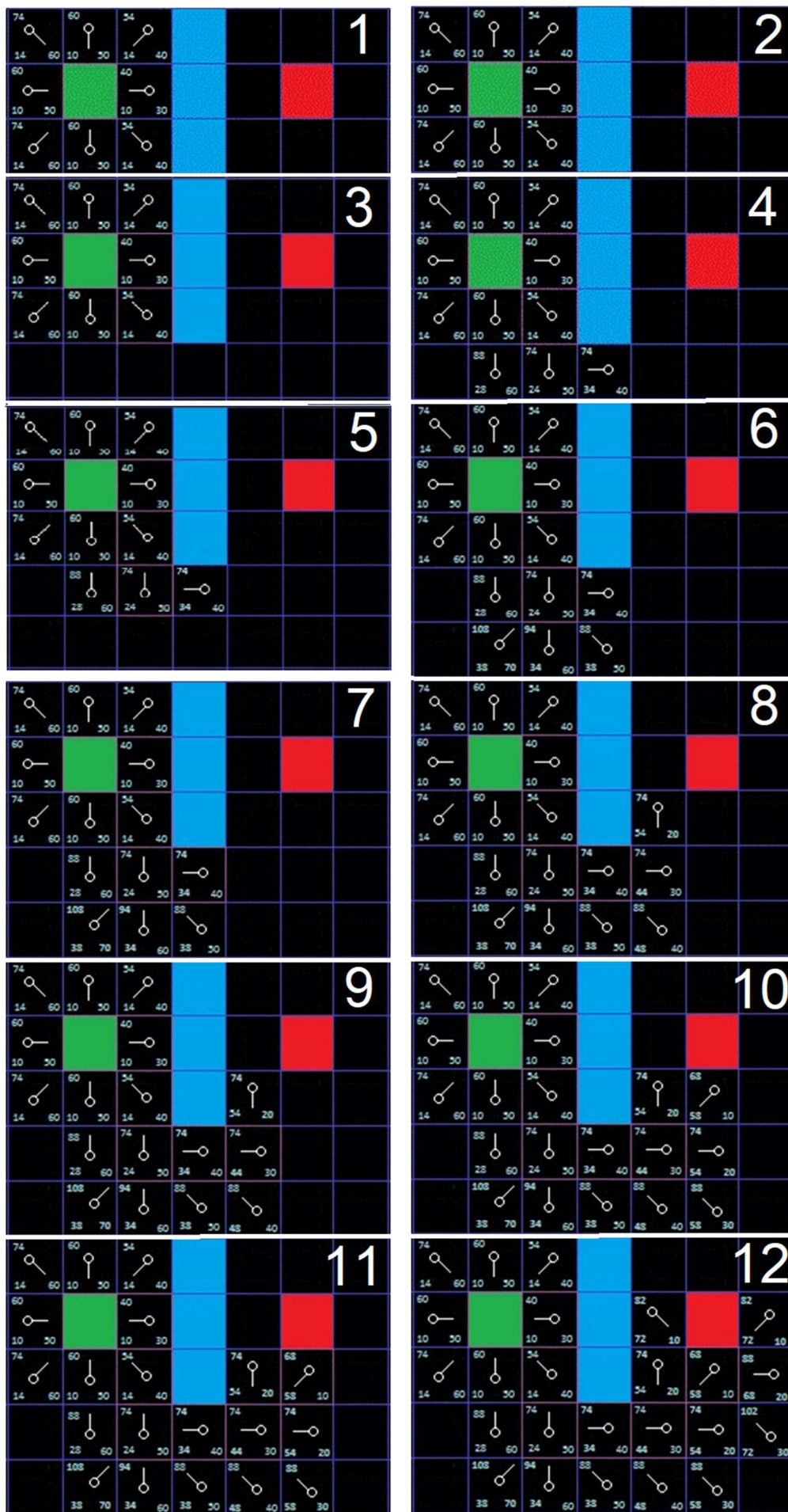


Рисунок 5.1 – Пример работы алгоритма.

## 6 ЭКСПЕРИМЕНТЫ

Для тестирования системы были проведены эксперименты. Условия экспериментов:

- машины добавлялись в случайное время, в случайном месте
- заранее заданы точки прибытия на «стоянке»
- необходимо объезжать стационарное препятствие

Алгоритм успешно справился с задачами:

- машины не врезаются друг в друга и в стационарные препятствия
- заранее останавливаются, чтобы пропустить проезжающие мимо машины, если подождать занимает меньше времени, чем объехать рядом едущие машины.

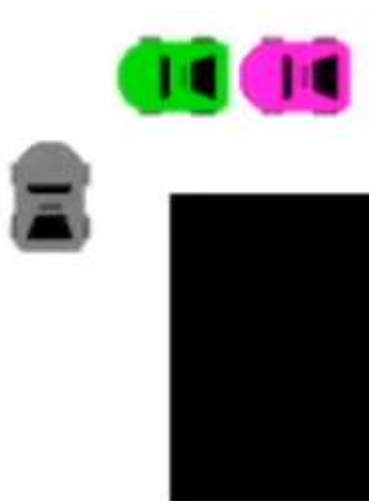


Рисунок 6.1 – Две машины остановились, чтобы пропустить серую машину

- машины чувствуют, что место, ранее занятое, освободилось. (могут проезжать по местам, которые ранее объезжали, так как они были заняты другими машинами)
- работают без светофоров (преимущество в том, что машины не стоят на светофорах большую часть времени)

Недоработки реализации алгоритма:

- машина №9 при движении из нижнего левого угла в верхний левый угол периодически поворачивает направо без необходимости



Рисунок 6.2 – Машина №9

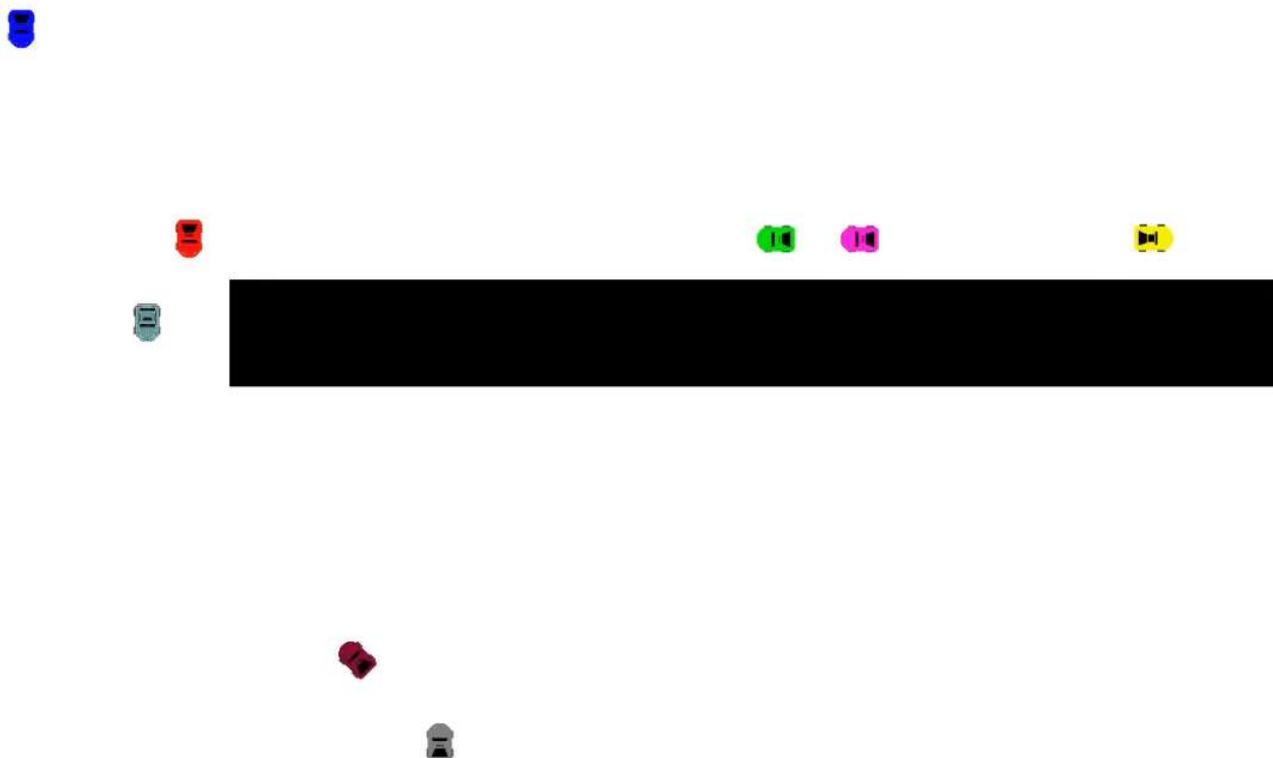


Рисунок 6.3 – 3 секунда работы симуляции

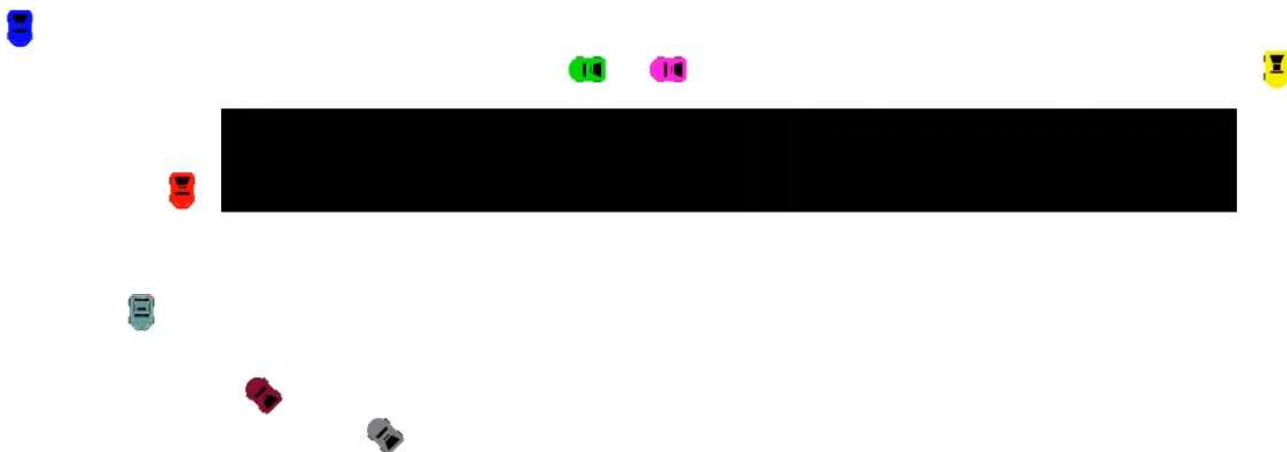


Рисунок 6.4 – 6 секунда работы симуляции



Рисунок 6.5 – 10 секунда работы симуляции



Рисунок 6.6 – 11 секунда работы симуляции



Рисунок 6.7 – 11 секунда работы симуляции



Рисунок 6.8 – 12 секунда работы симуляции



Рисунок 6.9 – 12 секунда работы симуляции



Рисунок 6.10 – 13 секунда работы симуляции

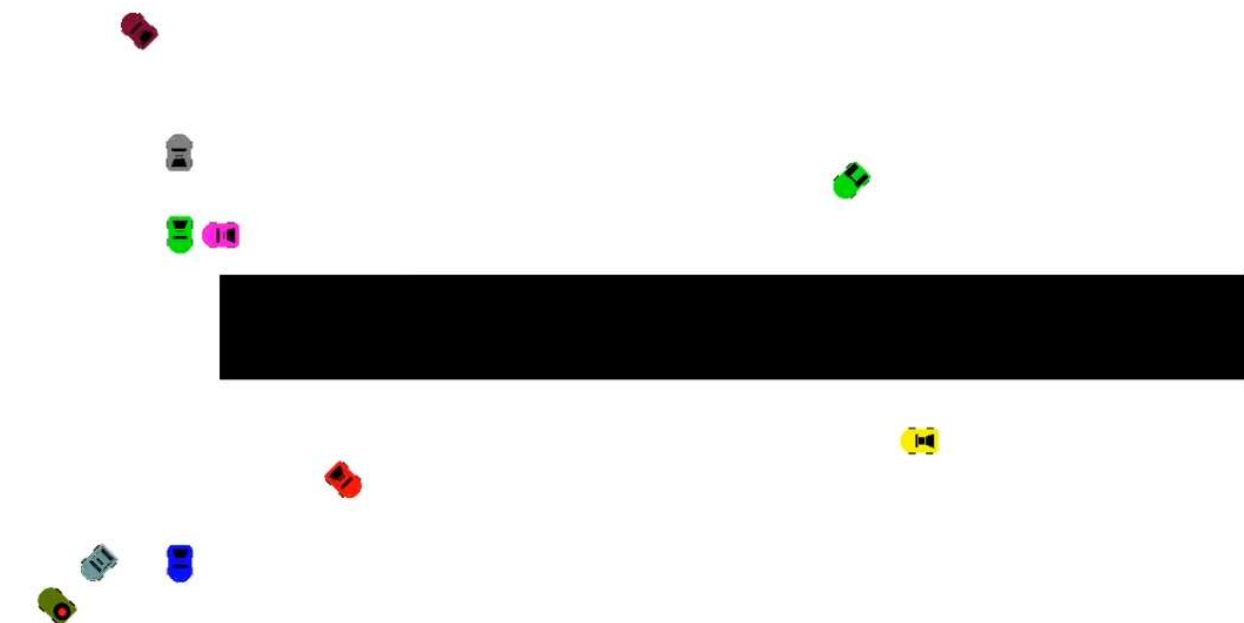


Рисунок 6.11 – 13 секунда работы симуляции



Рисунок 6.12 – 14 секунда работы симуляции

# ПРИЛОЖЕНИЕ А

(справочное)  
Библиография

- 1 Курносов М.Г. Введение в структуры и алгоритмы обработки данных. М. : Автограф, 2015. 12с.
- 2 Алгоритм поиска А\*. URL: <https://www.youtube.com/watch?v=AsEC2TJZ3JY> (Дата обращения: 11.05.2020)
- 3 Алгоритм Флойда. URL: <https://www.youtube.com/watch?v=HwK67u7zaEE> (Дата обращения: 11.05.2020)
- 4 Поиск пути в играх. Алгоритм поиска пути А\*. URL: <https://www.youtube.com/watch?v=gCclsViUeUk> (Дата обращения: 11.05.2020)
- 5 Ты знаешь откуда возникают пробки? URL: <https://www.youtube.com/watch?v=xwTMmdeLRKI> (Дата обращения: 11.05.2020)
- 6 Базовые алгоритмы нахождения кратчайших путей во взвешенных графах / Хабр. URL: <https://habr.com/ru/post/119158/> (Дата обращения: 11.05.2020)
- 7 GeorgiaFrankinStain/Centralized\_Control\_of\_Autopilots\_diploma: This is my diploma. URL: [https://github.com/GeorgiaFrankinStain/Centralized\\_Control\\_of\\_Autopilots\\_diploma](https://github.com/GeorgiaFrankinStain/Centralized_Control_of_Autopilots_diploma) (Дата обращения: 11.05.2020)
- 8 Видеозапись проведенного эксперимента с движением машин по кругу без препятствий URL: [https://vk.com/video2399234\\_136124429](https://vk.com/video2399234_136124429) (Дата обращения: 11.05.2020)



## ПРИЛОЖЕНИЕ Б

(рекомендуемое)

Наиболее употребляемые текстовые сокращения

ЦКА — центральный контроль  
автопилотов

ООП — объектно ориентированное  
программирование

UML - Unified Modeling Language

# ПРИЛОЖЕНИЕ В

## Листинги

### Листинг В.1 – Демонстрация иерархии файлов

```
src
  ConsoleManegement
    ConsoleManagement.java
  GUI
    ExecutionTaskRendering.BasicFeaturesJava
      RenderingFootprints
        Machine.java
        WallCar.java
        WallSquare.java
      DisplaingSpaces.java
      DisplaingSpacesClass.java
      FabricRenderingFootprintClass.java
      FabricRendringFootprint.java
      MapRender.java
      MapRenderClass.java
      RenderingFootprint.java
      TableObjectRendering.java
      TableObjectRenderingClass.java
      WindowsClass.java
    StatementTaskRendering
      ConvertersTime.java
      DemonstartionAlhorith.java
      MarginTimeForLevel.java
      MarginTimeForLevelClass.java
      ConverterTime.java
      DataFootprintForRendering.java
      HistChangesFromWhen.java
      PoolDataFootprintForRendering.java
      PoolDataFootprintForRenderingClass.java
      SubWindow.java
      TypeLandscapeBody.java
      TypeMachinesBody.java
      Windows.java
    UserCommandInterface
      UserCommandInterface.java
      UserCommandInterfaceClass.java
    Window
      AppearanceSettings.java
      GlobalAppearanceSettings.java
  Logic
    AreasBenchmarkPaths
      AreasBenchmarkPaths.java
      StraightLineEstimatedClass.java
    ControllerMachines
      AlhorithmFastFindPath.java
      AStarSpaceTime.java
      ControllerMachines.java
      ControllerMachinesClass.java
```

```

        NetworkNodes.java
        Node.java
        NodeClass.java
        SquareNetworkNodes.java
FootprintSpaceTime
    Exeption

CrashIntoAnImpassableObstacleExeption.java
    CreatorMarksOfPath.java
    CreatorMarksOfPathClass.java
    Footprint.java
    FootprintClass.java
    FootprintsSpaceTime.java
    FootprintsSpaceTimeClass.java
    LayerFootprintSpaceTime.java
    LayerFootprintSpaceTimeClass.java
    Point.java
    PointClass.java
    PolygonExtended.java
    PolygonExtendedClass.java
Landscape
    costil.java

CreatorLandscapeFromCommandInMachinesCostil.java
    Landscape.java
    LandscapeClass.java
    ZonaLandscape.java
    ZonaLandscapeClass.java
MovingObjects
    MovingObject.java
    MovingObjectClass.java
    Path.java
    PathClass.java
PathsMachines
    PositionClass.java
FabricMovingObjects.java
FabricMovingObjectsClass.java
GlobalVariable.java
LevelLayer.java
LevelLayerClass.java
Log.java
Position.java
TypesInLevel.java
PercistanceDataAccessObjects.java
Wrapper
    EntryPair.java
    EntryPairClass.java
    MyMultiMap.java
    MyMultiMapTree.java
    RandomWrapper.java
    RandomWrapperClass.java
Main.java
test
    GUI.StatementTaskRendering.ConvertersTime

```

```

        DemonstartionAlhorithTest.java
    Logic
        ControllerMachines
            AlhorithmFastFindPathTest.java
            NodeTest.java
        FootprintSpaceTime
            FootprintsSpaceTimeTest.java
            FootprintTest.java
            LocalToolRenderingPolygon.java
            PointTest.java
            PolygonExtendedTest.java
    Wrapper
        MyMultiMapTest.java
    SharedDevelop.java

```

## Листинг B.2 – FoorprintSpaceTime

```

package Logic.FootprintSpaceTime;

import Logic.MovingObjects.MovingObject;
import Logic.Position;

public interface Footprint {
    public int getIdObject();
    public int getIdTrack();
    public Position getPosition();
    public Point getCoordinat();
    public double getTimeStanding();
    public MovingObject getMovingObject();
    public void setTimeStanding(double newTimeStanding);
    public PolygonExtended getOccupiedLocation();
    //    public double getTravelTimeFromLastFootprint();

    public String toString();
    public boolean equals(Object obj);
}

```

## Листинг B.2 – FoorprintSpaceTimeClass

```

package Logic.FootprintSpaceTime;

import
Logic.FootprintSpaceTime.Exeption.CrashIntoAnImpassableOb
stacleExeption;
import Logic.LevelLayer;
import Logic.LevelLayerClass;
import Logic.MovingObjects.MovingObject;
import Logic.MovingObjects.Path;
import GUI.StatementTaskRendering.HistChangesFromWhen;
import Logic.Position;

```

```

import java.util.*;

public class FootprintsSpaceTimeClass implements
FootprintsSpaceTime, HistChangesFromWhen {

    Map<LevelLayer, LayerFootprintSpaceTime> layers =
        new TreeMap<LevelLayer,
LayerFootprintSpaceTime>();

    public FootprintsSpaceTimeClass() {
    }

    @Override
    public PolygonExtended getAreaChangesAfterBefore(int
afterTime, int berforeTime) {
        return null;
    }

    @Override
    public int getTimeLastUpdate() {
        return 0;
    }

    @Override
    public List<Footprint>
getRenderingFootprintsFromWhen(PolygonExtended areaFind,
double time, LevelLayer levelLayer) {
        return
this.layers.get(levelLayer).getRenderingFootprintsFromWhe
n(areaFind, time);
    }

    @Override
    public void addFootprint(
        int idTrack,
        MovingObject movingObject,
        Path path,
        double startTime,
        LevelLayer levelLayer
    ) throws CrashIntoAnImpassableObstacleExeption {
        LayerFootprintSpaceTime layerFootprintSpaceTime =
            this.layers.get(levelLayer);
        if (layerFootprintSpaceTime == null) {
            this.setLayer(levelLayer);
        }

        this.layers.get(levelLayer).addFootprint(
            idTrack,
            movingObject,
            path,

```

```

        startTime
    );
}

@Override
public void addFootprint(
    Footprint footprint,
    double time,
    LevelLayer levelLayer
) throws CrashIntoAnImpassableObstacleExeption {
    LayerFootprintSpaceTime layerFootprintSpaceTime =
        this.layers.get(levelLayer);
    if (layerFootprintSpaceTime == null) {
        this.setLayer(levelLayer);
    }

    this.layers.get(levelLayer).addFootprint(footprint,
time);
}

@Override
public void deleteFootprints(int ID) {

}

@Override
public boolean getIsSeatTaken(
    PolygonExtended place,
    double time,
    LevelLayer levelLayer
) {

    LayerFootprintSpaceTime layerFootprintSpaceTime =
        this.layers.get(levelLayer);
    if (layerFootprintSpaceTime == null) {
        return false;
    }

    return
layerFootprintSpaceTime.getIsSeatTaken(place, time);
}

@Override
public boolean getIsSeatTakenSpaceTime(PolygonExtended
place, double fromTime, double toTime, LevelLayer
levelLayer) {
    return this.getIsSeatTaken(place, fromTime,
levelLayer) //FIXME
        && this.getIsSeatTaken(place,
toTime, levelLayer)

```

```

        && this.getIsSeatTaken(place,
(fromTime + toTime) / 2, levelLayer);
    }

    @Override
    public Double averageTimeMovingToNextPointOfPath() {
        for(Map.Entry entry: this.layers.entrySet()) {
            LayerFootprintSpaceTime value =
(LayerFootprintSpaceTime) entry.getValue();
            Double localResult =
value.getAverageTimeMovingToNextPointOfPath();
            if (localResult != null) {
                return localResult;
            }
        }

        return null;
    }

    @Override
    public Double getTimeAddingLastFootprints(LevelLayer
levelLayer) {
        LayerFootprintSpaceTime layerFootprintSpaceTime =
this.layers.get(new LevelLayerClass(0));
        if (layerFootprintSpaceTime == null) {
            return null;
        }

        return
layerFootprintSpaceTime.getTimeAddingLastFootprints();
    }

    @Override
    public Position getPositionInDefaultLevel(int ID,
double time) {
        //FIXME one machine with some ID in one moment time
        LayerFootprintSpaceTime layerFootprintSpaceTime =
this.layers.get(new LevelLayerClass(0));
        if (layerFootprintSpaceTime == null) {
            return null;
        }

        return layerFootprintSpaceTime.getPosition(ID,
time);
    }

    //==== <start> <Private_Methods>
    =====
    private void setLayer(LevelLayer levelLayer) { //FIXME
CODESTYLE
        LayerFootprintSpaceTime layerFootprintSpaceTime =

```

```

        new LayerFootprintSpaceTimeClass();
        this.layers.put(levelLayer,
layerFootprintSpaceTime);
    }
    //====<end> <Private_Methods>
=====
=====
}

```

### Листинг В.3 – LayerFootprintSpaceTimeClass

```

package Logic.FootprintSpaceTime;

import
Logic.FootprintSpaceTime.Exeption.CrashIntoAnImpassableOb
stacleExeption;
import Logic.GlobalVariable;
import Logic.Landscape.Landscape;
import Logic.LevelLayer;
import Logic.MovingObjects.MovingObject;
import Logic.MovingObjects.Path;
import Logic.Position;
import Logic.TypesInLevel;
import Wrapper.EntryPair;
import Wrapper.MyMultiMap;
import Wrapper.MyMultiMapTree;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class LayerFootprintSpaceTimeClass implements
LayerFootprintSpaceTime {

    private MyMultiMap<Double, Footprint>
storageAllFootprints =
        new MyMultiMapTree<Double, Footprint>();

    // ListMultimap<Double, Footprint> storageAllFootprints
= ArrayListMultimap.create();

    // private Map<Double, List<Footprint>>
storageAllFootprints = new TreeMap<Double,
List<Footprint>>();
    // private List<MovingObject> imitationLandscape = new
ArrayList<MovingObject>(); //FIXME IMITATION Landscape
    /*
        program min:
            polygons
        program max:
            rounds
    */

```



```

*/

public LayerFootprintSpaceTimeClass() {
}

@Override
public void addFootprint(
    int idTrack,
    MovingObject movingObject,
    Position position,
    double time,
    double timeStanding
) throws CrashIntoAnImpassableObstacleExeption {
//FIXME ADD_TEST

    Footprint newFootprint = new
FootprintClass(idTrack, position, timeStanding,
movingObject);

    this.addFootprint(newFootprint, time);
}

@Override
public void addFootprint(Footprint footprint, double
time) throws CrashIntoAnImpassableObstacleExeption {

    boolean placeIsSeat =
this.getIsSeatTaken(footprint.getOccupiedLocation(),
time);

    if (!placeIsSeat) {
        storageAllFootprints.put(time, footprint);
    } else {
        throw new
CrashIntoAnImpassableObstacleExeption();
    }
}

@Override
public void addFootprint(
    int idTrack,
    MovingObject movingObject,
    Path path,
    double startTime
) throws CrashIntoAnImpassableObstacleExeption {
    CreatorMarksOfPath creatorMarksOfPath =
        new CreatorMarksOfPathClass(this, idTrack,
movingObject);
    creatorMarksOfPath.addFootprint(path, startTime);
}

```

```

        @Override
        public void deleteFootprints(int ID) {
            //FIXME
        }

        @Override
        public boolean getIsSeatTaken(PolygonExtended place,
double testedTime) { //FIXME ADD_TEST

//FIXME NOW
            Iterator<EntryPair<Double, Footprint>>
iteratorEntryPair =
storageAllFootprints.iteratorEntryPair();
            while (iteratorEntryPair.hasNext()) {
                EntryPair<Double, Footprint> entry =
iteratorEntryPair.next();
                Footprint footprint = entry.getValue();
                double timeStandingStart = entry.getKey();
                double timeStandingEnd = timeStandingStart +
footprint.getTimeStanding();
                boolean timeStandingIncludeTestedTime =
timeStandingStart < testedTime && testedTime <
timeStandingEnd;

                if (timeStandingIncludeTestedTime) {
                    PolygonExtended locationMovingObject =
footprint.getOccupiedLocation();
                    boolean placeIsSeat =
place.intersectionPolygon(locationMovingObject);
                    if (placeIsSeat) {
                        return true;
                    }
                }
            }

            return false;
        }

        @Override
        public Position getPosition(int ID, double time) {
//FIXME NOW
            Iterator<EntryPair<Double, Footprint>>
iteratorEntryPair =
storageAllFootprints.iteratorEntryPair();
            while (iteratorEntryPair.hasNext()) {
                EntryPair<Double, Footprint> entry =
iteratorEntryPair.next();
                Footprint currentFootprint = entry.getValue();
                double startStanding = entry.getKey();
                double endStanding = startStanding +
currentFootprint.getTimeStanding();

```

```

        boolean    timeStandingIncludeFindTime    =
startStanding <= time && time < endStanding;
        boolean    isFindObject    =
currentFootprint.getIdObject()    ==    ID    &&
timeStandingIncludeFindTime;
        if (isFindObject) {
            return currentFootprint.getPosition();
        }
    }

    return null;
}

@Override
public Double getAverageTimeMovingToNextPointOfPath()
{
    Double averageRes = null;

    Iterator<EntryPair<Double, Footprint>>
iteratorEntryPair    =
storageAllFootprints.iteratorEntryPair();
    while (iteratorEntryPair.hasNext()) {
        EntryPair<Double, Footprint>    entry    =
iteratorEntryPair.next();
        Footprint footprint = entry.getValue();
        boolean    isEndOfPath    =
GlobalVariable.equalsNumber(
            footprint.getTimeStanding(),

CreatorMarksOfPathClass.MAX_TIME_STANDING
        );
        if (!isEndOfPath) {
            if (averageRes == null) {
                averageRes    =
footprint.getTimeStanding();
            } else {
                averageRes    +=
footprint.getTimeStanding();
                averageRes /= 2;
            }
        }
    }

    return averageRes;
}

@Override
public Double getTimeAddingLastFootprints() { //FIXME
ADD_TEST
    Double lastTime = Double.MIN_VALUE;
    Iterator<EntryPair<Double, Footprint>>
iteratorEntryPair    =
storageAllFootprints.iteratorEntryPair();
    while (iteratorEntryPair.hasNext()) {

```

```

        EntryPair<Double, Footprint> entry =
iteratorEntryPair.next();
        Double timeAdding = entry.getKey();
        if (timeAdding > lastTime) {
            lastTime = timeAdding;
        }
    }

    return lastTime;
}

//TODO: add more difficult determitaion the level
(https://habr.com/ru/post/122919/)
//TODO: return id of poligons returned getAreaFromWhen
используется выделителем юнитов, тут не требуется
возвращать полигоны, можно просто айдишники вернуть
@Override
public List<Footprint>
getRenderingFootprintsFromWhen(PolygonExtended
areaVizibility, double timeFind) {

    //FIXME take DataFootprintForRendering from the
    landscape

    //iteration all polygons
    // add in resList, if intersection with
    areaVizibility

    /* program min:
        return a list of all polygons from a
    intersection table with areaVizibility (so far everything
        is stored in a single table)

    program max:
        return a list of changed polygons from a
    intersection table with areaVizibility*/
    // ArrayList newList = (ArrayList)
    this.imitationLandscape;
    //TODO add interpolation (LINK_RzRGrmTH)

    List<Footprint> resRendringFootpring = new
    ArrayList<Footprint>();
    Iterator<EntryPair<Double, Footprint>>
    iteratorEntryPair =
    storageAllFootprints.iteratorEntryPair();
    while (iteratorEntryPair.hasNext()) {
        EntryPair<Double, Footprint> entry =
    iteratorEntryPair.next();

        Footprint currentFootprint = entry.getValue();
    //FIXME NOW add test timeFind diapason intersection

```

```

        double          timeStanding          =
currentFootprint.getTimeStanding();
        double startStanding = entry.getKey();
        double  endStanding  =  startStanding  +
timeStanding;

        boolean  footprintIndcludeFindTimePoint  =
startStanding <= timeFind && timeFind < endStanding;
        if (footprintIndcludeFindTimePoint) {

resRendringFootpring.add(currentFootprint);
        }
    }

    return resRendringFootpring;
}

}

```

#### Листинг В.4 – CreatorMarksOfPathClass

```

package Logic.FootprintSpaceTime;

import
Logic.FootprintSpaceTime.Exeption.CrashIntoAnImpassableOb
stacleExeption;
import Logic.GlobalVariable;
import Logic.MovingObjects.MovingObject;
import Logic.MovingObjects.Path;
import Logic.PathsMachines.PositionClass;
import Logic.Position;

public class CreatorMarksOfPathClass implements
CreatorMarksOfPath {
    final public static double MAX_TIME_STANDING =
Double.MAX_VALUE * 0.95;
    private LayerFootprintSpaceTime footprintsSpaceTime;
    private int idTrack;
    private MovingObject movingObject;

    private double speed;
    private double lengthStep;
    double timeStanding;
    private Footprint penultimateFootprintInPath = null;
    private Footprint lastFootprintInPath = null;

    public CreatorMarksOfPathClass(

```

```

        LayerFootprintSpaceTime footprintsSpaceTime,
        int idTrack,
        MovingObject movingObject
    ) {
        this.footprintsSpaceTime = footprintsSpaceTime;
        this.idTrack = idTrack;
        this.movingObject = movingObject;

        /**
         *                               Landscape                               have
        getResistancePowerLandscape (pressurePaskaleOfMachine)
         * speed = MachinePower / ResistancePower
         */
        speed = movingObject.getSpeed();
        lengthStep = movingObject.getLength();
        if (Math.abs(speed) <
GlobalVariable.DOUBLE_COMPARISON_ACCURACY) {
            timeStanding = this.MAX_TIME_STANDING;
        } else {
            timeStanding = lengthStep / speed;
        }
    }

    @Override
    public void addFootprint(
        Path path,
        double startTime
    ) throws CrashIntoAnImpassableObstacleExeption {

        try {
            addFootprinsBasedOnThePath(path, startTime);
        } catch (CrashIntoAnImpassableObstacleExeption ex)
        {

            setTheStandingTimeUntilTheEndOfTimeInCaseOfAnAccident();
            throw new
            CrashIntoAnImpassableObstacleExeption();
        }

        }

        //====<start>                               <Private_Methods>
        =====
        =====
        private void
        setTheStandingTimeUntilTheEndOfTimeInCaseOfAnAccident() {
            if (this.penultimateFootprintInPath != null) {

                this.penultimateFootprintInPath.setTimeStanding(this.MAX_
                TIME_STANDING);
            }
        }
        private void addFootprinsBasedOnThePath(

```

```

        Path path,
        double startTime
    ) throws CrashIntoAnImpassableObstacleExeption {

        double timeAdding = startTime;

        if (path.getSize() == 0) {
            assert (false);
        } else if (path.getSize() == 1) {
            Position position = new
PositionClass(path.getPoint(0), 0.0);
            this.addFootprint(
                position,
                timeAdding,
                this.MAX_TIME_STANDING
            );
        } else {
            timeAdding +=
processingCreateFootprintsOnRouteStraightLineFromPairPoin
ts(path, timeAdding);
            //FIXME ADD_TEST BAG += equals on result = (add
sumTime equal on result adding positionTime)

processingCreateFoorprintEndRouteFromSinglePoint(path,
timeAdding);
        }
    }

    private double
processingCreateFootprintsOnRouteStraightLineFromPairPoin
ts(
        Path path,
        double timeAdding
    ) throws CrashIntoAnImpassableObstacleExeption {
        double sumTime = 0;
        int endIndex = path.getSize() - 1;
        for (int i = 0; i < endIndex; i++) {
            Point startLine = path.getPoint(i);
            Point endLine = path.getPoint(i + 1);

            double lastSumTime = printEveryStepOnLine(
                startLine,
                endLine,
                timeStanding,
                timeAdding
            );

            sumTime += lastSumTime;
            timeAdding += lastSumTime;
        }

        return sumTime;
    }
}

```

```

/**
 *
 * number this is number iteration cicle up
 * number this is point
 * -- this is time standing (Length equal length
movingObject)
 * - this is lastLittleStep (last Little Time Standing)
 * 0--0--0--0-1
 *
 *      |
 *      |
 *      1
 *      |
 *      |
 *      1
 *      |
 *      2--2--2--2-3
 *
 *                      ^
 *                      end point in endless
 *
 *
 * @param path
 * @param timeAdding
 * @throws CrashIntoAnImpassableObstacleExeption
 */
private void
processingCreateFoorprintEndRouteFromSinglePoint(
    Path path,
    double timeAdding
) throws CrashIntoAnImpassableObstacleExeption {
    int indexLastPoint = path.getSize() - 1;
    Point startLine = path.getPoint(indexLastPoint -
1);
    Point endlessPoint =
path.getPoint(indexLastPoint);
    double angleStepVector =
endlessPoint.getAngleRotareRelative(startLine); //FIXME
dublication (LINK_RletVeVp)
    Position position =
PositionClass(endlessPoint, angleStepVector);
    this.addFootprint(
        position,
        timeAdding,
        this.MAX_TIME_STANDING
    );
}

private boolean stepUnderTest(Point startLine, Point
endLine, Point stepUnderTest) {
    int quarterStartLine =
startLine.getQuarter(endLine);
    int quarterstepUnderTest =
stepUnderTest.getQuarter(endLine);

```



```

        return quarterStartLine == quarterstepUnderTest;
    }

    private Point stepVector(Point endLine, Point
startLine, double lengthStep) {
        Point origin = new PointClass(0, 0);
        double angleStepVector =
endLine.getAngleRotareRelative(startLine);
        return new PointClass(lengthStep,
0).getRotareRelative(origin, angleStepVector); //FIXME
MAGIC NUMBER
    }

    private double printEveryStepOnLine(
        Point startLine,
        Point endLine,
        double standingTime,
        double timeAdding
    ) throws CrashIntoAnImpassableObstacleExeption {

        double angle =
endLine.getAngleRotareRelative(startLine);
        double timeSum = 0;

        Point currentCoordinat = startLine.clone();
        Point stepVector = stepVector(endLine, startLine,
movingObject.getLength());

        double angleStepVector =
endLine.getAngleRotareRelative(startLine);

        double lengthStep = stepVector.getLengthVector();
        double lengthStraightPath =
endLine.getDistanceToPoint(startLine);
        int counterMaxSteps = (int) (lengthStraightPath /
lengthStep);

        for (int i = 0; (i < counterMaxSteps) &&
this.stepUnderTest(startLine, endLine, currentCoordinat);
i++) {

            Position position = new
PositionClass(currentCoordinat, angleStepVector);
            double sumTimeAddFootprint =
this.addFootprint(
                position,
                timeAdding,
                standingTime
            );

            timeAdding += standingTime +
sumTimeAddFootprint; //FIXME TECHNICAL CREDIT
            timeSum += standingTime + sumTimeAddFootprint;

```

```

        currentCoordinat = new PointClass(
            currentCoordinat.getX() +
stepVector.getX(),
            currentCoordinat.getY() +
stepVector.getY()
        );
    }

    timeSum +=
littleStepInEndStraightLineBetweenTwoNeighborePointPaths
(
    endLine,
    currentCoordinat,
    angleStepVector,
    timeAdding
);

    return timeSum;
}

private double
littleStepInEndStraightLineBetweenTwoNeighborePointPaths
(
    Point endLine,
    Point currentCoordinat,
    double angleStepVector,
    double timeAdding
) throws CrashIntoAnImpassableObstacleExeption {
    double localTimeSum = 0;
    if (!endLine.equals(currentCoordinat)) {

        /*
        * |--|--| -
        *      ^
        * little step*/
        Point penultimatePoint = currentCoordinat;
        double lengthFinalStep =
endLine.getDistanceToPoint(penultimatePoint);
        double standingTime = lengthFinalStep /
movingObject.getSpeed();

        Position position = new
PositionClass(penultimatePoint, angleStepVector);
        localTimeSum = standingTime;
        this.addFootprint(
            position,
            timeAdding,
            standingTime
        );
    }
}

```

```

        return localTimeSum;
    }

    return localTimeSum;
}

private double addFootprint( //FIXME must type void
TECHNICAL CREDIT
    Position position,
    double time,
    double timeStanding
) throws CrashIntoAnImpassableObstacleExeption {

    double addSum = 0;

    while (true) {
        boolean susseful = true;
        try {
            this.penultimateFootprintInPath =
this.lastFootprintInPath;
            this.lastFootprintInPath = new
FootprintClass(idTrack, position, timeStanding,
movingObject);

this.footprintsSpaceTime.addFootprint(this.lastFootprintI
nPath, time);
        } catch (CrashIntoAnImpassableObstacleExeption
ex) {
            susseful = false;
        }

        if (susseful) {
            break;
        } else {
            time += timeStanding;
            addSum += timeStanding;
        }

    }

    return addSum;
}

//==== <end> <Private_Methods>
=====
=====
}

```