**РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ**

# Факультет физико-математических и естественных наук Кафедра прикладной информатики и теории вероятностей

**ОТЧЕТ**

**ПО УЧЕБНОЙ ПРАКТИКЕ**

Студент: Косолапов Степан Эдуардович

Группа: НПИбд-01-20

# МОСКВА

2023 г.

# Постановка задачи

Установить и изучить среду для моделирования сетей VANET – SUMO. Установить ns3 и приобрести навыки работы с ns3.

# Выполнение работы

# Установка SUMO

# Выполнять установку будем на операционной системе macOS.

# Установим sumo через менеджера пакетов brew. Для этого переходим на официальный сайт SUMO и проходим по указанным шагам: <https://sumo.dlr.de/docs/Installing/index.html#macos>.

# Обновляем brew: brew update

# Устанавливаем XQuartz: brew install --cask xquartz

# Устанавливаем SUMO: brew tap dlr-ts/sumo brew install sumo

# Добавляем в файл конфигурации shell(мы используем zsh) переменную окружения SUMO\_HOME: Проверяем, что переменная добавилась: Text Description automatically generated

# Далее установим графический интерфейс для sumo – sumo-gui. Делаем это так же через менеджера пакетов brew: brew install --cask sumo-gui

# Теперь надо перезапустить операционную систему.

# После этого нам доступно приложение sumo-gui. Открываем его: Graphical user interface, text, application, Word Description automatically generated

# Симуляция в SUMO

# Для ознакомления попробуем запустить какую-нибудь симуляцию. Есть много способов создать файл симуляции, но самый быстрый для нас сейчас – это использовать [Web Wizard](https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html). Это программа, позволяющая нам создать файл симуляции через графический интерфейс. При этом мы можем просимулировать практически любое место на карте. Web Wizard использует [openstreetmap](https://www.openstreetmap.org/) для создания симуляции.

# Чтобы запустить Web Wizard – откроем официальную документацию и пройдем по шагам: <https://sumo.dlr.de/docs/Tutorials/OSMWebWizard.html>

# Нам нужно запустить файл osmWebWizard.py. Он лежит в директории tools. Graphical user interface, application Description automatically generated Запускаем этот файл через интерпретатор python: Map Description automatically generated

# Открывается графический интерфейс. Здесь мы можем выбрать интересующие нас параметры. Например город и количество автомобилей на дорогах: Graphical user interface, application Description automatically generated Graphical user interface, text, application Description automatically generated

# Нажимаем generate scenario и файл с сгенерированной сетью открывается в sumo: Graphical user interface, diagram Description automatically generated

# В консоли снизу мы видим откуда был открыт файл. Text Description automatically generated

# Для интереса мы можем его изучить: Мы видим, что это файл xml. В его начале есть какая-то конфигурация: Text Description automatically generated Далее идет само описание сгенерированной сети: Text Description automatically generated Из понятного – мы видим тут ребра, которые отображаются на карте. Так же можно разобрать, что на нашей карте будут перекрёстки – они описаны в тэгах junction Text Description automatically generated

# Попробуем запустить теперь эту симуляцию. Для этого нажимаем кнопку Run в SUMO: Graphical user interface, text Description automatically generated

# Мы сразу увидим, как по городу начнут передвигаться желтые автомобили: Graphical user interface Description automatically generated with medium confidence

# Мы можем уменьшить скорость их передвижения с помощью изменения параметра Delay в большую сторону:

# Мы так же можем выполнять симуляцию пошагово кнопкой Step: Graphical user interface Description automatically generated

# Экспорт трейса из SUMO

# В этапе 2 мы рассматривали создание файла .sumocfg через WebWizard. Это файл конфигурации, который описывает сценарий нашей симуляции. Мы можем прогнать этот сценарий через sumo и на выходе получить файл, который будет описывать состояние системы в каждый промежуток времени. Чтобы узнать как это сделать, мы можем обратиться к официальной документации - <https://sumo.dlr.de/docs/Tutorials/Trace_File_Generation.html>.

# Пройдем по шагам, указанным в документации: Запускаем sumo, передавая на вход файл нашего сценария. В качестве output указываем, что нам нужен fcd файл, то есть floating car data файл(см [документацию](https://sumo.dlr.de/docs/Simulation/Output/FCDOutput.html)), и название выходного файла, назовем его sumoTrace.xml: Text Description automatically generated with medium confidence

# Видим логи запуска симуляции: Тут есть кое-какая статистика по итогам симуляции. Из интересного, видим, что в программе, видимо, был использован алгоритм Дейкстры для поиска кратчайшего пути. В документации мы можем найти подтверждение этой догадке - <https://sumo.dlr.de/docs/Simulation/Routing.html#routing_algorithms> В документации так же говорится, что мы даже можем выбрать astar вместо алгоритма Дейкстры, если нам это нужно, просто указав опцию --routing-algorithm Text, letter Description automatically generated

# После выполнения сценария мы получаем на выходе xml файл с трейсом:

# Взглянем на получившийся файл: В нем мы видим набор тэгов timestep, которые содержат в себе состояние системы в определенный промежуток времени. Текущий промежуток времени определяется аттрибутом time. В нашем случае – внутри timestep присутствуют тэги, обозначающие состояние автомобиля. Каждый автомобиль имеет свой уникальный идентификатор, координаты, тип, скорость и другие параметры.

# Graphical user interface, text, application Description automatically generated Можно заметить, что с течением времени старые автомобили пропадают(выезжают за карту), а новые автомобили добавляются: Text Description automatically generated

# Установка ns-3

# Теперь нам нужно установить ns-3. Для начала скачаем с [официального сайта](https://www.nsnam.org/releases/ns-3-38/download/) архив с кодом: Graphical user interface, text, application Description automatically generatedРаспаковываем архив и видим ряд файлов, Text Description automatically generated

# Теперь мы можем запустить файл build.py: Стоит заметить, что предварительно требуется установить clang и gcc, cmake и python. Началась компиляция файлов: Text Description automatically generated

# После установки заходим в директорию ns-3.38.

# В ней лежат все скомпилированные файлы, они готовы к использованию:

# Мы можем провалидировать установку, запустив команду ./test.py: Text Description automatically generated

# Все скомпилировалось успешно:

# Теперь мы можем запустить скомпилированный ns3. Так как мы указали –enable-examples при компиляции, то у нас есть ряд примеров, которые мы можем запустить. Например, запустим hello-simulator: Graphical user interface Description automatically generated with low confidence

# Видим, что все корректно работает.

# Преобразование трейса в файл .tsl

# Теперь, когда у нас есть трейс, мы можем преобразовать его в тот формат, который мы можем использовать в ns3. Например .tsl. Чтобы узнать как это сделать – смотрим официальную документацию sumo - <https://sumo.dlr.de/docs/Tools/TraceExporter.html#ns2ns3>

# Пройдем по шагам из документации. Нужно запустить скрипт traceExporter.py из папки tools: Text Description automatically generated

# После выполнения скрипта – видим наш .tcl файл:

# Открыв этот файл, мы видим список нод: Тут после at указан timestep. Для каждого timestep указана строка, применяющая позицию(команда setdest) к каждому узлу. Как мы и ожидали, у нас столько же timestep как и в трейсе, а так же мы видим, что последний узел – 711, а значит общее количество узлов 712, как было в трейсе. То есть файл сконвертировался корректно: Graphical user interface, text Description automatically generated

# Теперь мы можем использовать этот файл в программе для ns3

# Вывод

Мы изучили графический интерфейс среды для симуляции SUMO. А также попробовали запустить простую симуляцию. Установили ns3.