**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по учебной практике**

**ТЕМА: «Алгоритм A*»**

Студент гр. 9382 _____ Савельев И.С.

Студентка гр. 9382 _____ Круглова В.Д.

Студент гр. 9303 _____ Дюков В.А.

Руководитель _____ Ефремов М.А.

Санкт-Петербург

2021

**ЗАДАНИЕ**

**НА УЧЕБНУЮ ПРАКТИКУ**

Студент Савельев И.С. группы 9382

Студентка Круглова В.Д. группы 9382

Студент Дюков В.А. группы 9303

Тема практики: алгоритма A*

Задание на практику: разработка визуализатора алгоритма A* на Java.

Сроки прохождения практики: 01.07.2021 – 14.07.2021

Дата сдачи отчета: 14.07.2021

Дата защиты отчета: 14.07.2021

| | | |
|---|---|---|
| Студент | _____ | Савельев И.С |
| Студентка | _____ | Круглова В.Д. |
| Студент | _____ | Дюков В.А. |
| Руководитель | _____ | Ефремов М.А. |

## АННОТАЦИЯ

Целью учебной практики является разработка приложения для визуализации алгоритма A*. Приложение создается на языке Java и должно обладать графическим интерфейсом. Пользователю должна быть предоставлена возможность заполнения матрицы (включая стартовую и конечную точку, препятствия), а также пошагового выполнения алгоритма с пояснениями. Приложение должно быть понятным и удобным для использования.

Задание выполняется командой из трех человек, за которыми закреплены определенные роли. Выполнение работы и составление отчета осуществляются поэтапно.



## SUMMARY

The purpose of the practice is to develop an application for visualizing the A * algorithm. The application is created in Java and must have a graphical interface. The user should be given the opportunity to fill out the matrix (including the start and end points, obstacles), as well as step-by-step execution of the algorithm with explanations. The application should be clear and easy to use.

The task is carried out by a team of three people, for which certain roles are assigned. The execution of the work and the preparation of the report are carried out in stages.

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Целью учебной практики является создание приложения, визуализирующего работу алгоритма A*, предназначенного для нахождения кратчайших расстояний между двумя точками. Приложение должно быть написано на языке Java и снабжено понятным и удобным в использовании графическим интерфейсом. Пользователю должна быть предоставлена возможность ввести исходные данные в самой программе с клавиатуры. Результат работы алгоритма также должен выводиться на экран. Должна быть предоставлена возможность как моментального отображения результата, так и визуализации пошагового выполнения алгоритма.

Задание выполняется командой из трех человек, за каждым из которых закреплены определенные обязанности – реализация графического интерфейса, логики алгоритма, проведение тестирование и сборка проекта. В ходе сборки должны выполняться модульные тесты и завершаться успехом. Также на момент завершения практики должен быть составлен подробный отчет, содержащий моделирование программы, описание алгоритмов и структур данных, план тестирования, исходный код и др.

# 1. ТРЕБОВАНИЯ К ПРОГРАММЕ

## 1.1. Требования к вводу исходных данных

Исходными данными для реализуемого приложения является матрица, в которой будет осуществляться поиск путей. Пользователю предлагается заполнить матрицу, используя следующие обозначения: 1 – преграда, E/e – конечная точка, S/s – стартовая точка.

## 1.2. Требования к выводу результата

Результат выполнения алгоритма должен выводиться на экран в виде матрицы, чьи ячейки окрашены в определенные цвета. Красный – вершина принадлежит закрытому списку, зелёный – вершина принадлежит открытому списку, пурпурный – в этом месте преграда.

## 1.3. Требования к визуализации

Необходимо реализовать удобный и понятный пользователю графический интерфейс. Должна быть предоставлена возможность отрисовки заданной матрицы, выполнение алгоритма по требованию пользователя необходимо осуществлять моментально с выводом результата или пошагово. При пошаговом выполнении алгоритма каждый этап должен быть снабжен пояснениями.

# 2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

## 2.1. План разработки

К 03.07.2021 должны быть распределены роли между членами бригады, создан репозиторий проекта и изучены основы Java.

К 06.07.2021 реализован прототип интерфейса программы, реализован прототип алгоритма A*, проработана архитектура приложения. Написан отчет с UseCase и Class диаграммами.

К 08.07.2021 реализован базовый функционал приложения, реализована связь между функционалом программы и интерфейсом.

К 10.07.2021 расширены возможности работы с приложением, с точки зрения взаимодействия с интерфейсом, убраны критические ошибки.

К 12.07.2021 реализована финальная версия программы, произведено тестирование и отладка программного кода. Написана финальная версия отчета. Проект готов к сдаче.

## 2.2. Распределение ролей в бригаде

Савельев И.С. front-end разработчик.

Круглова В.Д. back-end разработчик.

Дюков В.А. тестировщик.

# 3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

## 3.1. Структуры данных

Алгоритм A*, реализуемый в программе, предназначен для обработки матрицы. Хранение матрицы осуществляется при помощи класса **JTable**. Матрица включает в себя массив клеток, к которым можно получить доступ с помощью метода **getValueAt**. Класс **AStar** состоит из полей, необходимых для выполнения всех методов класса, например, список открытых и список закрытых вершин. Класс **Pair** хранит координаты точки на матрице.

Объект класса **AStar** создается при нажатии на кнопки *Start*, *Next*, *ToEnd* и хранится в виде указателя в классе **GUI** программы, в котором реализован графический интерфейс. Чтение исходных данных для создания графа может осуществляется с помощью ввода на экране. Вывод результата обработки графа также осуществляется на экран. Эти классы осуществляют взаимосвязь между пользовательским интерфейсом и графом.

**Реализованные классы.**

**`public class`** `AStar` – класс самого алгоритма A*, в котором находятся методы и обьекты для реализации алгоритма

**`private`** `Pair minF(List<Pair> open, Map<Pair, Float> F)` – метод класса Astar для поиска следующей ячейки для посещения, получает на вход список открытых ячеек и значения их расстояний необходимых для рассчета.

**`private boolean`** `inList(List<Pair> _list, Pair point)` – метод класса Astar , который определяет входит ли данная пара в список переданный методу.

**`private void`** `repaint(JTable table, Pair _pair, String color_type)` – метод класса Astar , который перекрашивает выбранную ячейку

в определенный цвет.

**private void** `reconstruction(JTable table, Map<Pair, Pair> _map)` – метод класса Astar , который вызывается после нахождения алгоритмом конечной точки. Метод перекрашивает и помечает получившийся кратчайший путь от начальной до конечной точки.

**private void** `ReturnToAlgo(JTable table, Map<Pair, Pair> _map)` – метод класса Astar , который позволяет из финального положения с построенным и отмеченым путем вернуться в режим построения пути. Метод перекрашивает финальный путь в закрытые вершины.

**public int** `nextStep(JTable table)` – метод класса Astar , который получает на вход таблицу и совершает один шаг алгоритма : Ищет новые свободные вершины, Выбирает из них самую оптимальную, Переходит в нее. При этом отображая все это цветом.

**public int** `backStep(JTable table)` – метод класса Astar , обратный методу `nextStep` , то есть возвращается на предыдущий шаг алгоритма, возвращая цветовую прорисовку с прошлого шага.

**public** `Pair getCurrPosition()` – метод класса Astar , который возвращает текущее положение алгоритма в таблице.

**class** `Pair` – это класс, который содержит в себе два обьекта примитивного типа.

**class** `MyRenderer` **extends** `DefaultTableCellRenderer` – это класс, наследуемый от `DefaultTableCellRenderer`.Этот класс позволяет перекрашивать ячейки таблицы `JTable` в другой цвет.

### 3.2. Описание алгоритма

Алгоритм A* реализован в методе *nextStep* класса **AStar.** Исходный код алгоритма представлен в приложении А. На каждом шаге метод изменяет матрицу, которая содержит значения открытых и закрытых вершин, пройденного пути, обрабатываемые с помощью покраски в определенный цвет.

Алгоритм содержит два цикла, в которых обходятся все клетки матрицы, значения которых сравниваются со значениями из списка кратчайших путей и текущий кратчайший путь из одной вершины в другую сравнивается с суммой путей из начальной клетки в текущую. Если путь через обрабатываемую клетку короче текущего пути, в списке кратчайших путей изменяется кратчайший путь между клетками.

### 3.3. Алгоритм пошагового отображения

Пошаговое выполнение алгоритма представляет собой, по сути, одну итерацию основного алгоритма. Для его реализации было решено хранить переменные, использующиеся для обхода матрицы в циклах алгоритма, в качестве полей класса. Переменные предварительно проверяются на невыход за границы матрицы.

Далее выполняется шаг алгоритма, т. е. текущий кратчайший путь из одной клетки в другую сравнивается с путем, проходящим через соседнюю клетку. Если путь через соседнюю клетку короче текущего пути, в матрице кратчайших путей изменяется кратчайший путь между клетками.

### 3.3. Основные методы

Методы алгоритма реализованы в классе **AStar**, такие как реконструкция финального пути, методы, соответствующие кнопкам *Back, Next, toStart, toEnd, Start.*

## 4. ИНТЕРФЕЙС

### 4.1. Реализация интерфейса

Взаимодействие пользователя с программой осуществляется при помощи графического интерфейса (рисунок 1). При запуске программы открывается главное окно программы, состоящее из текущего состояния таблицы, окна для ввода данных, описания легенды матрицы и управляющих кнопок.

Нажатием на кнопки пользователь может выполнить шаг алгоритма вперёд и назад, перейти в начальное и конечное состояния.

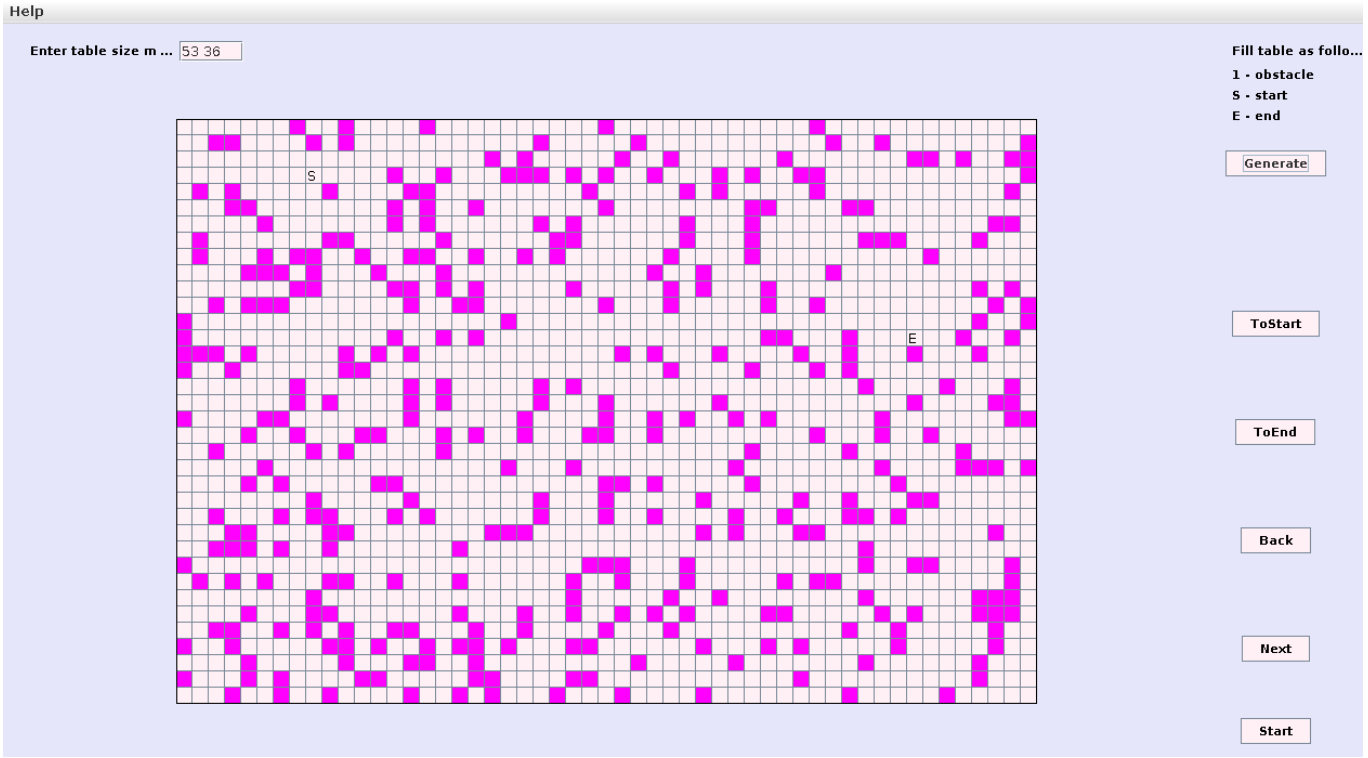



Рисунок 1 - Главное окно программы

При вводе новой матрицы она отрисовывается на экране. В матрице пользователь может поставить начальную и конечную клетку алгоритма.

При запуске алгоритма или шаге алгоритма отрисовывается матрица результата, содержащая текущие кратчайшие пути между вершинами графа. Желтым цветом отображаются клетки кратчайшего пути. Белым не используемые в ходе алгоритма. Зеленым отображаются клетки помещенные в открытый список. Пурпурным непроходимые клетки. Красным клетки помещенные в закрытый список.

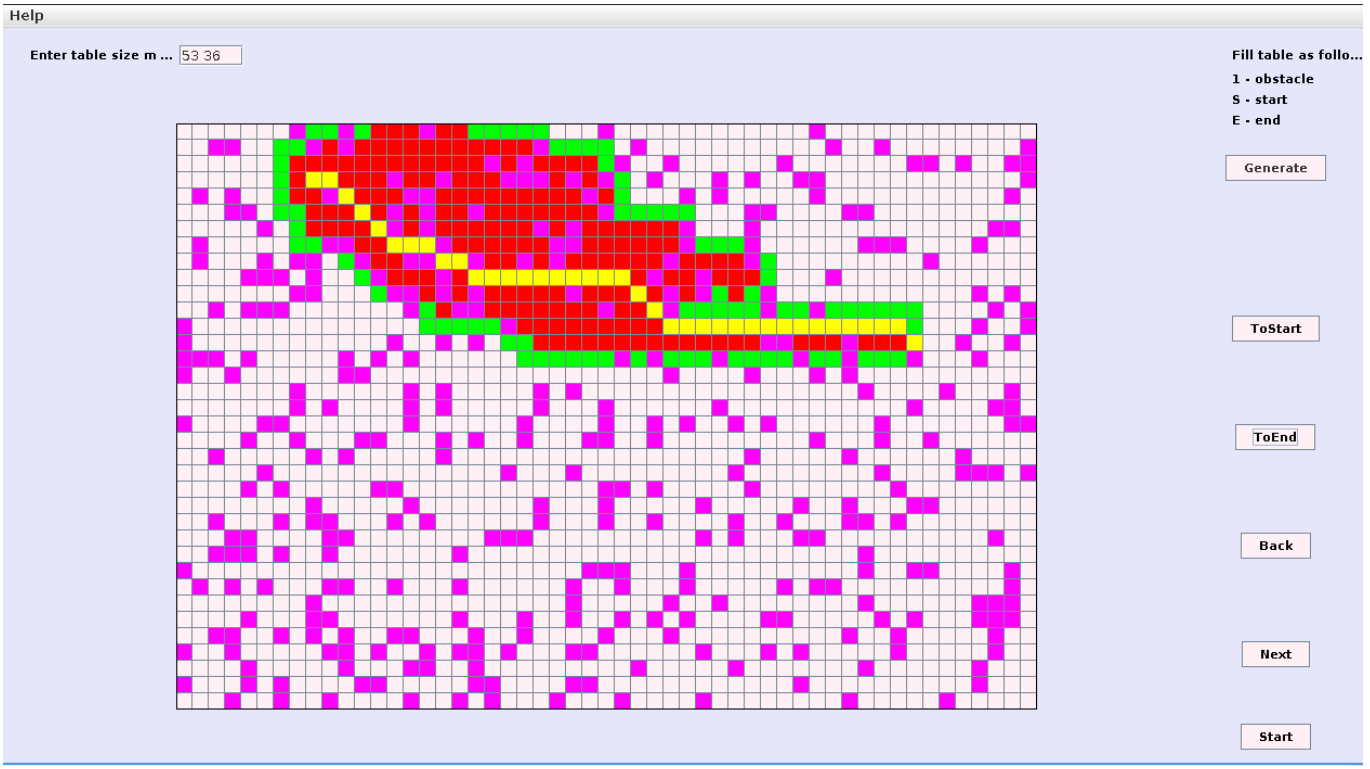
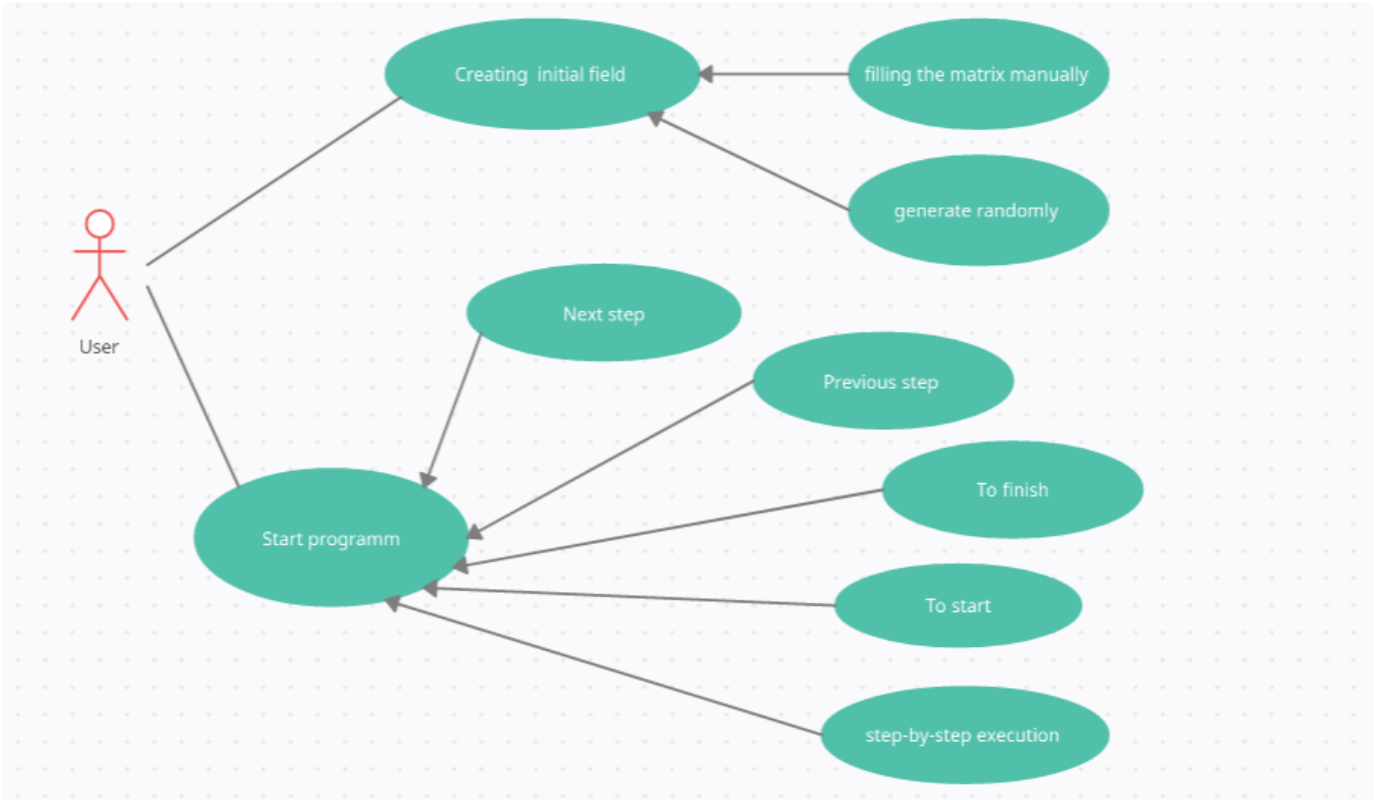



Рисунок 2 - Результат работы программы

## 4.2. Use case диаграмма



# ЗАКЛЮЧЕНИЕ

# ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА