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 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 граф, в котором будет осуществляться поиск путей. Пользователю предлагается заполнить его, используя следующие обозначения: 1 – преграда, E/e – конечная точка, S/s – стартовая точка или сгенерировать его автоматически.

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 заданного графа, 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. реализация графического интерфейса.

Круглова В.Д. тестирование программы, написание отчета.

Дюков В.А. реализация внутренней логики программы.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Для хранения графа и его компонентов, в программе были реализованы следующие классы:

1. Point – класс, представляющий вершину графа. Хранит поля с координатами:
    - int x;
    - int y;

Поля отвечают за ее идентификацию и неизменяемы. Их значения можно получить с помощью методов getX() и getY().

Для копирования вершины используется соответствующий конструктор в классе.

2. Graph – основной класс, представляющий граф. Хранит единственное поле:
    - HashMap<Point, HashMap<Point, Integer>> map;

Хэш таблица позволяет одновременно хранить все вершины графа, а также получать данные о соседях, с которыми соединена текущая вершина и вес инцидентных ребер. Ключом на первом уровне таблицы является вершина, из которой выходит ребро, на втором – вершина, в которую входит ребро. Значения во внутренней таблице – длины ребер графа.

Реализованные методы данного класса представляют функционал для добавления и удаления вершин и ребер, а также для получения состояния полей графа.

Для копирования графа используется соответствующий конструктор в классе.

3.2. Описание основных классов для реализации логики приложения.

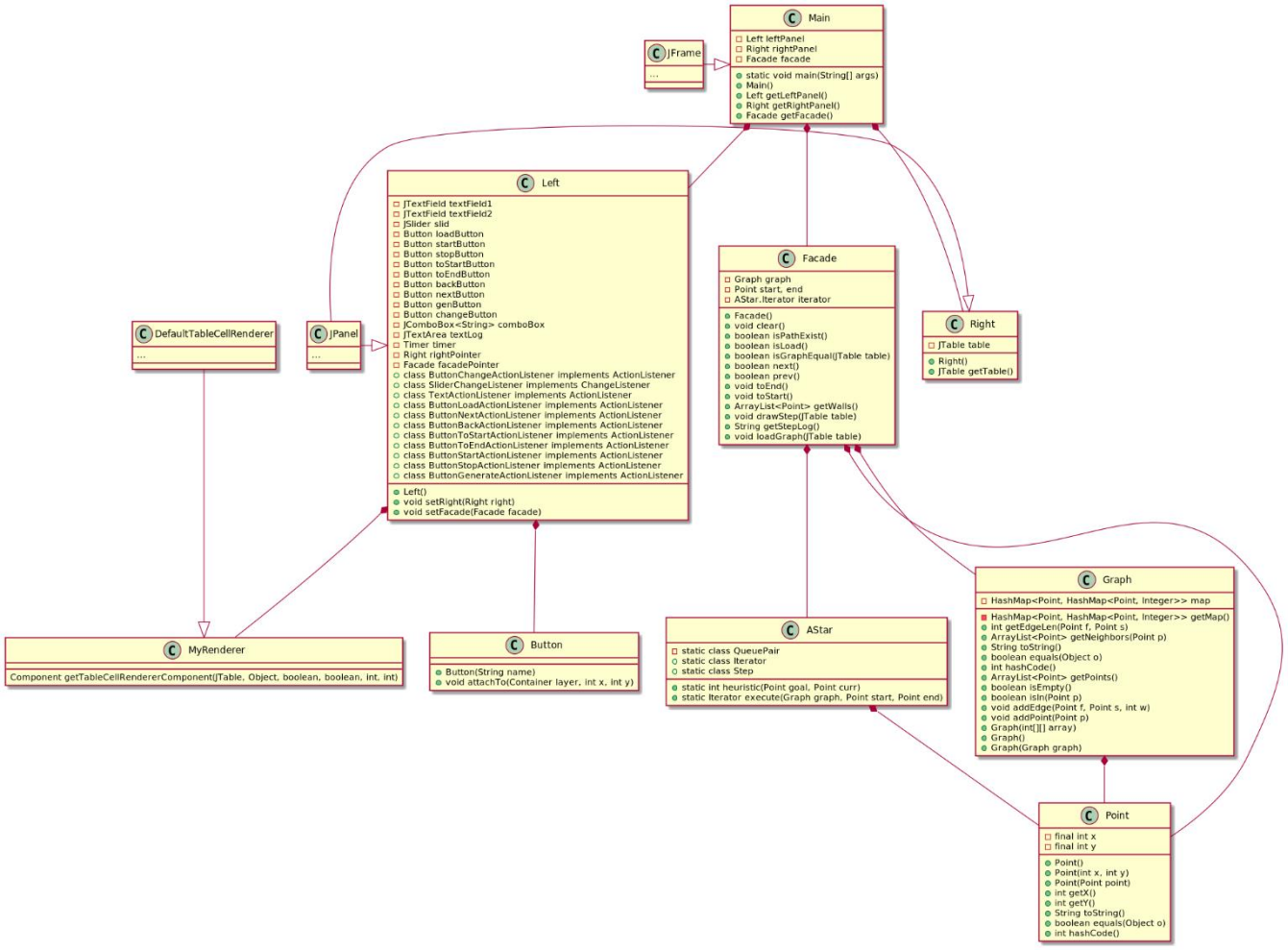
Для реализации функционала приложения были реализованы следующие классы:

- Graph, Point – классы, для хранения графа.
- AStar – класс, содержащий методы, для реализации алгоритма A*, а также дополнительные классы, необходимые для реализации методов, а также использующиеся в других частях программы.
- QueuePair – закрытый внутренний класс AStar, представляющий собой пару число-вершина. Используется в приоритетной очереди, в реализации метода execute() класса AStar.
- Iterator – внутренний класс A*, представляющий собой итератор по массиву объектов класса Step, и используется при пошаговом отображении алгоритма.
- Step – внутренний класс A*, хранящий конкретный шаг выполнения алгоритма. Используется при графическом отображении алгоритма A*.
- Facade – класс, обобщающий имеющуюся логику, и предоставляющий методы, удобные при реализации графического интерфейса.

Для реализации выбранной архитектуры приложения использовались следующие паттерны:

- Фасад – для обобщения всех классов (Graph, Point, AStar, Iterator), которые реализуют логику программы, с целью упрощения ее использования.
- Итератор – для удобной итерации всех шагов в алгоритме A*.

Ниже для конкретного понимания реализованных классов и их взаимосвязи представлена UML диаграмма классов.



UML диаграмма классов

3.3. Описание алгоритма.

На каждом шаге выбирается вершина с наименьшим приоритетом. Функция для оценки приоритета состоит из двух слагаемых: текущего расстояния от начальной вершины и эвристической функции (сумма расстояний по двум координатам между двумя вершинами).

Затем для выбранной вершины рассматриваются смежные ей вершины. Для каждой смежной вершины проверяется ее кратчайший путь до начальной вершины, и, если текущий путь короче, то заменяется на него. После этого смежная вершина помещается в очередь с приоритетом, где значение приоритетности определяется с помощью функции оценки приоритетности, определенной выше.

Алгоритм заканчивает работу, как только из очереди выйдет конечная вершина.

3.4. Описание реализации пошагового отображения.

Для отображения пошаговой работы алгоритма использовался паттерн Итератор. В реализующем его классе были определены следующие методы:

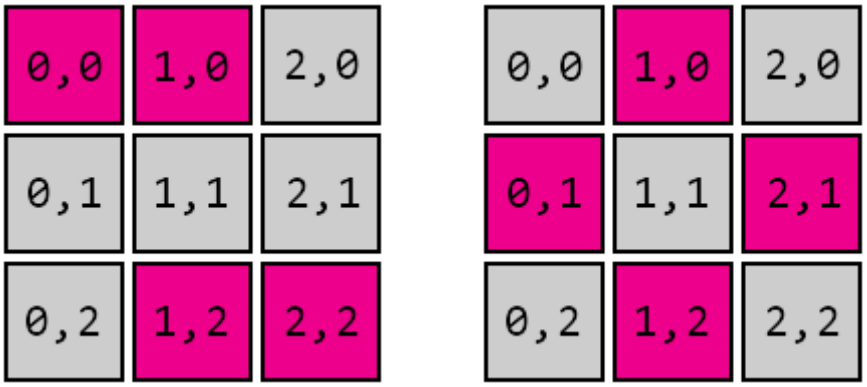
- Step next() – метод, перемещающийся на следующий элемент итератора. Возвращает этот элемент.
- Step prev() – метод, перемещающийся на предыдущий элемент итератора. Возвращает этот элемент.
- Step curr() – метод, возвращающий текущий элемент итератора.
- Step toStart() – метод, устанавливающий итератор на начало. Возвращает первый элемент.
- Step toEnd() – метод, устанавливающий итератор на конец. Возвращает последний элемент.

Итератор генерируется при вызове метода execute() класса Astar. Он содержит данные о каждом шаге алгоритма, которые и возвращает в методах.

Управление над Итератором осуществляется объектом класса Facade.

4. ТЕСТИРОВАНИЕ

Для проверки правильности реализованной логики работы алгоритма был реализован набор тестов, который представлен в таблице ниже. При тестировании использовались матричные графы размера 3 на 3. G1 представлен на рис. 1, а. G2 и G4 представлены на рис. 1, б. Граф G3 – пустой.



а б

Рисунок 1 – Тестируемые графы, а – граф G1, б – граф G2 и G4

Таблица 1 – Результаты тестирования

Тестовая задача	Входные данные	Результат
Проверка на корректность инициализации графа с помощью двумерного массива целых чисел. (Graph(int[][] array))	G1	done

Проверка на корректность инициализации графа с помощью двумерного массива целых чисел. (Graph(int[][] array))	G2	done
Проверка графов на неэквивалентность. (equals(Object o))	G1, G2	done
Проверка графов на эквивалентность. (equals(Object o))	G2, G4	done
Проверка графа на пустоту. (isEmpty())	G3	done
Проверка графа на непустоту. (isEmpty())	G1	done
Проверка графа на нахождение вершины в графе. (isIn(Point p))	G1, Point(1, 1)	done
Проверка графа на отсутствие вершины в графе. (isIn(Point p))	G1, Point(6, 1)	done
Проверка на отсутствие ребра в графе. (getEdgeLen(Point f, Point s, int w))	G1, Point(0, 0), Point(0, 1)	done

Проверка на существование ребра в графе. (getEdgeLen(Point f, Point s, int w))	G1, Point(1, 1), Point(0, 1)	done
Проверка поиска соседних вершин в графе. (getNeighbors(Point p))	G1, Point(1, 1)	done
Проверка отсутствия соседних вершин в графе. (getNeighbors(Point p))	G2, Point(1, 1)	done
Проверка поиска соседних вершин в графе, при вершине, отсутствующей в графе. (getNeighbors(Point p))	G2, Point(0, 1)	done

5. ИНТЕРФЕЙС

5.1. 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, окон для ввода данных, описания легенды матрицы, управляющих кнопок и слайдера.

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

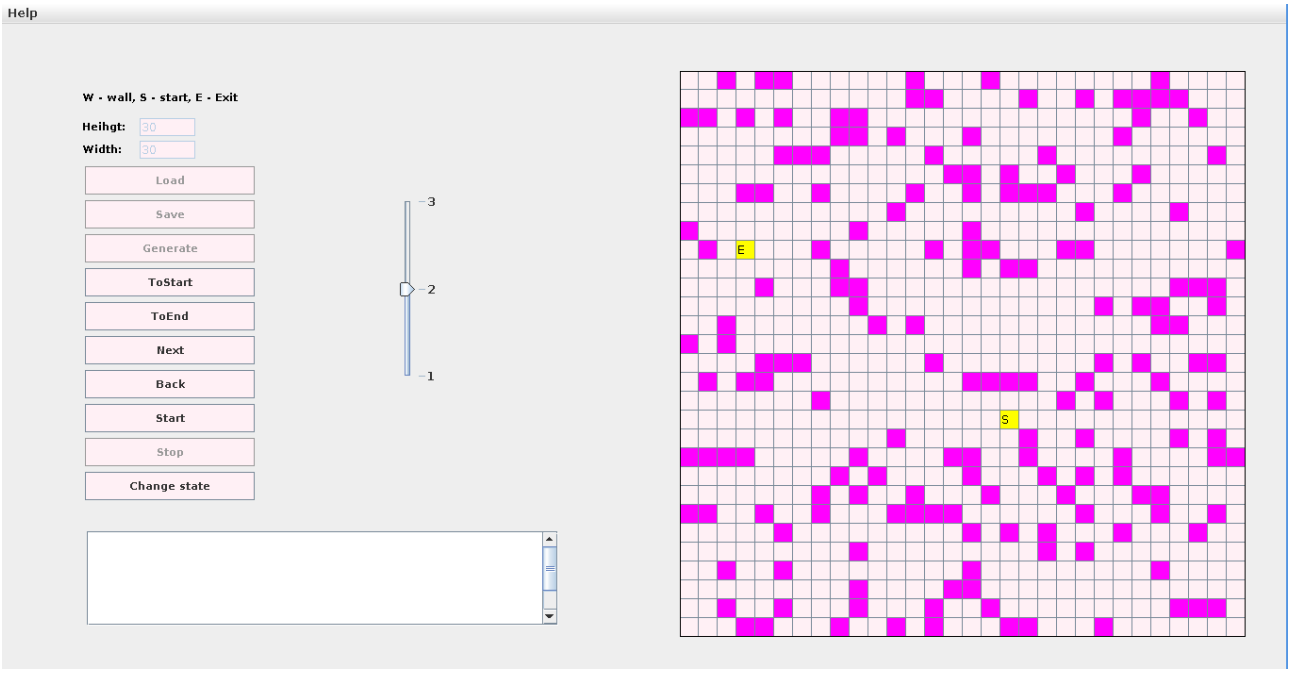


Рисунок 1 - Главное окно программы

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.

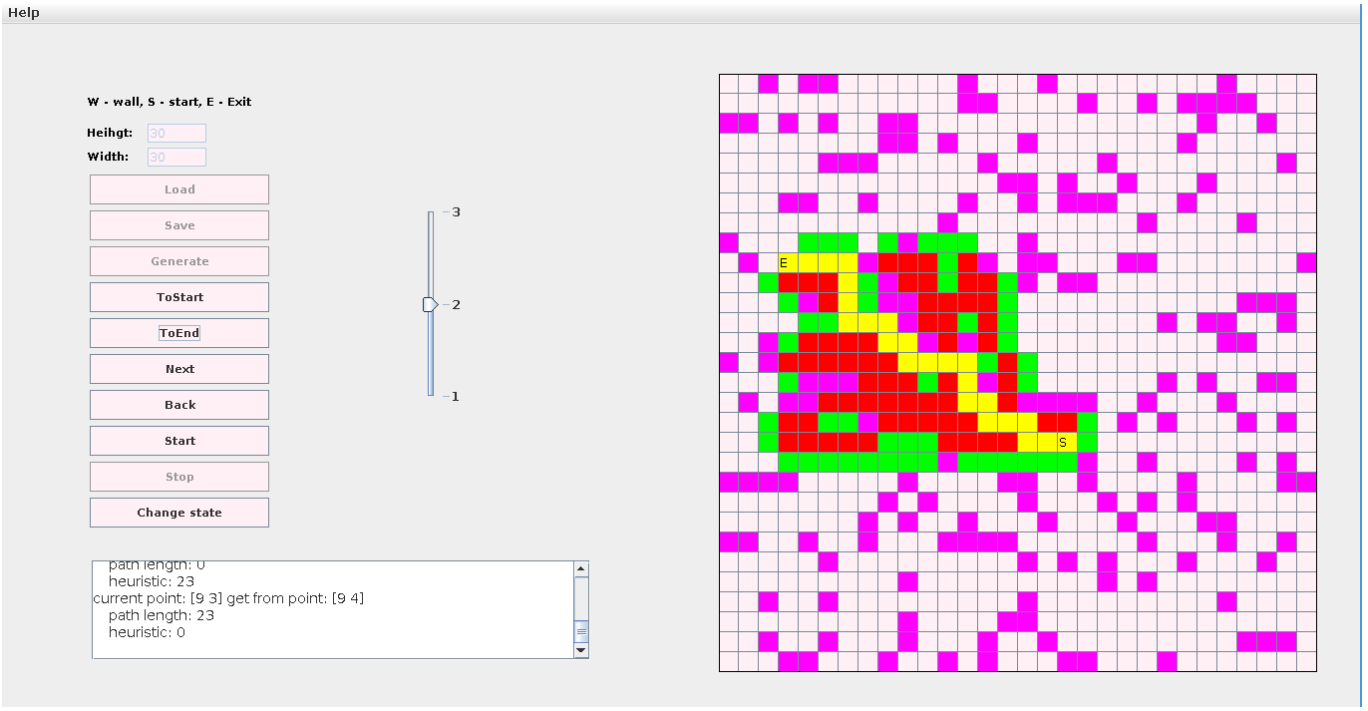
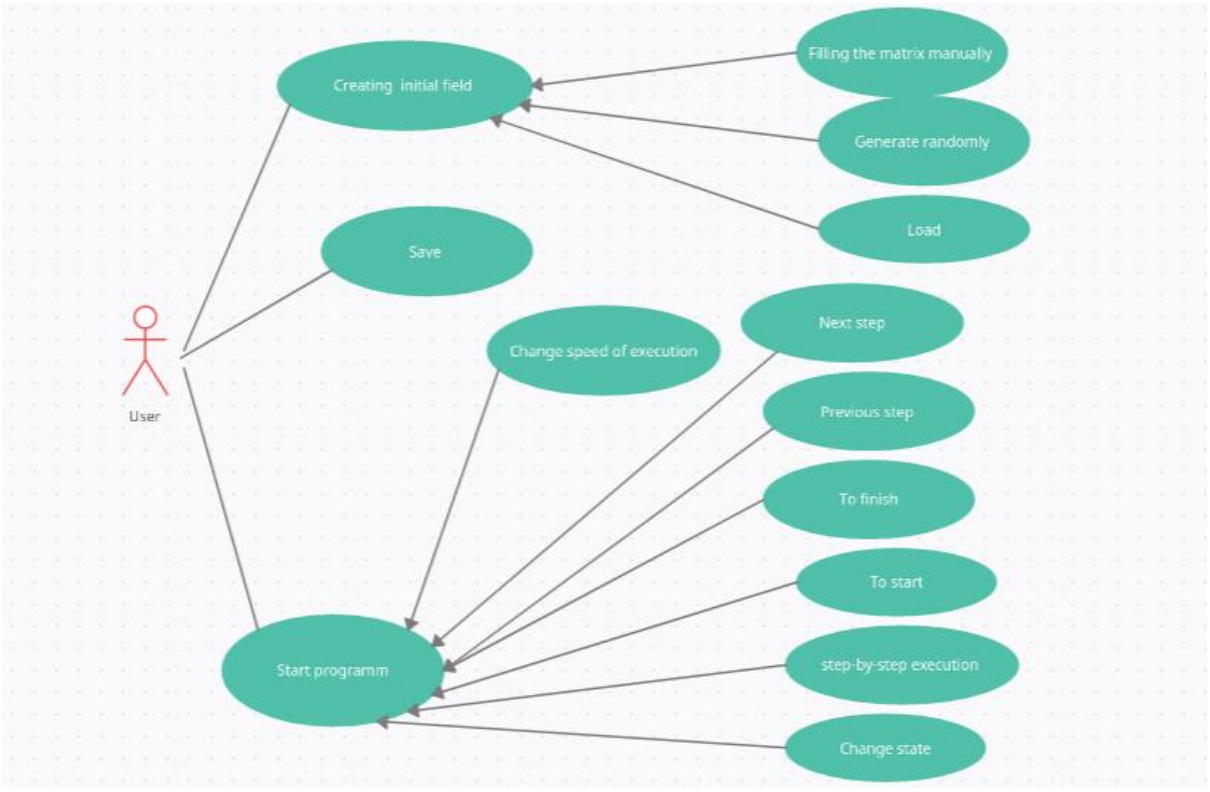


Рисунок 2 - Результат работы программы

5.2. Use case диаграмма



Для взаимодействия пользователя с интерфейсом были реализованы следующие элементы GUI:

- Два текстовых поля для ввода высоты и ширины поля, стены, начальную и конечную точку на котором пользователь может выбрать по своему усмотрению.
- Кнопка для случайной генерации поля.
- Кнопка для загрузки поля из файла.
- Кнопка для сохранения текущего поля в файл.
- Кнопка ToEnd - перемещает алгоритм к последнему шагу, отображая финальный путь, раскрашивая в соответствующие цвета клетки находящиеся в закрытом и открытом списке.
- Кнопка ToStar - перемещает алгоритм к исходному состоянию.
- Кнопки Next и Back позволяют пользователю пошагово пройти весь алгоритм, Next передвигает алгоритм на один шаг вперед, Back на один шаг назад, соответствующие изменения отображаются на поле.
- Кнопка Start запускает автоматическое выполнение алгоритм, которое можно остановить с помощью кнопки Stop, скорость автоматического выполнения алгоритма можно регулировать с помощью слайдера.
- Кнопка Change state останавливает выполнение алгоритма и позволяет изменить поле, например добавить стены к текущему полю или сгенерировать/загрузить совершенно новое.

ЗАКЛЮЧЕНИЕ

В ходе работы были выполнены следующие задачи:

- Были распределены роли участников команды, определена цель и план работы. Для большего понимания основ Java всеми участниками команды был пройден соответствующий электронный курс. Также было налажено взаимодействие членов команды (используя репозиторий GitHub). Однако стоит отметить, что во время ведения проекта, из-за неучтенных обстоятельств (малого опыта работы в команде), команда испытывала трудности в синхронизации работы.
- Была создана и согласована спецификация программы, подготовлен эскиз окна приложения а так же USE-CASE и UML диаграммы.
- Было создано графическое приложение, удовлетворяющее большинству требований, которые были оговорены на этапе согласования спецификации программы.
- Были исправлены критические баги при работе приложения, реализован дополнительный функционал (покраска компонент связности в разные цвета).
- Было проведено тестирование программы (а именно логики), для проверки правильности реализации логики приложения.
- Стоит отметить, что не всегда получалось выполнять все требования в срок, ввиду ограниченности (сроков), не всегда удачной коммуникации членов команды и отсутствия опыта использования Java ранее.

ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА

1. Шилдт, Г., 2015. Java8. Полное руководство. Издательский дом "Вильямс",

2. Эккель, Б., 2018. Философия Java. Издательство "Питер".

3. Хорстман, К., 2019. Java. Библиотека профессионала, том 1. ООО "Диалектика".

4. Блох, Дж., 2014. Java. Эффективное программирование. Издательство "Лори"