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

ОТЧЕТ

по учебной практике

Тема: Реализация алгоритма A* и визуализация её работы на языке Java

Студент гр. 1384 _____ Овчинников М.Ю.

Студент гр. 1384 _____ Сочков И.С.

Руководитель _____ Шестопалов Р.П.

Санкт-Петербург
2023

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Овчинников М.Ю. группы 1384

Студент Сочков И.С. группы 1384

Тема практики: Реализация алгоритма A* и визуализация её работы на языке Java

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: A* (A star).

Сроки прохождения практики: 30.06.2023 – 13.07.2023

Дата сдачи отчета: 13.07.2023

Дата защиты отчета: 13.07.2023

Студент _____ Овчинников М.Ю.

Студент _____ Сочков И.С.

Руководитель _____ Шестопалов Р.П.

АННОТАЦИЯ

Цель данной практики заключается в реализации алгоритма A* и визуализации его работы на языке Java. Алгоритм A* является эффективным поисковым алгоритмом, используемым для нахождения кратчайшего пути в графе с учетом оценок стоимости переходов. Основное содержание практики включает создание классов для представления графа и его вершин, реализацию алгоритма A* с использованием приоритетной очереди и определением эвристической функции, а также разработку графического интерфейса пользователя для визуализации процесса работы алгоритма. Визуализация будет включать отображение графа, выделение пройденных путей и показ текущего состояния поиска.

SUMMARY

The purpose of this practice is to implement the A* algorithm and visualize its operation in Java. Algorithm A* is an efficient search algorithm used to find the shortest path in a graph, taking into account estimates of the cost of transitions. The main content of the practice includes the creation of classes to represent the graph and its vertices, the implementation of the A* algorithm using a priority queue and the definition of a heuristic function, as well as the development of a graphical user interface for visualizing the process of the algorithm. Visualization will include displaying a graph, highlighting the paths traveled and showing the current search status.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Изучение языка программирования Java является важной задачей для студентов и разработчиков, так как этот язык широко используется в различных сферах программирования. Одной из ключевых применений Java является разработка алгоритмов и приложений для решения сложных задач. В рамках данной практики ставится цель овладеть навыками реализации алгоритма A* на языке Java и визуализации его работы.

Цель практики:

Целью данной практики является овладение навыками программирования на языке Java и реализация алгоритма A* с визуализацией. Алгоритм A* является одним из наиболее популярных поисковых алгоритмов, широко применяемых в различных областях, таких как искусственный интеллект, компьютерные игры, робототехника и планирование маршрутов. Он позволяет находить оптимальный путь в графе, учитывая стоимость переходов и ограничения.

Задачи практики:

В рамках данной практики будут решены следующие задачи:

Изучение основ языка программирования Java; знакомство с основными концепциями, синтаксисом и структурой языка Java. Это позволит освоить необходимые инструменты для реализации алгоритма A* и его визуализации.

Разработка классов для представления графа и его вершин. Это позволит удобно оперировать графом и его элементами.

Реализация алгоритма A* с использованием приоритетной очереди и эвристической функции для оценки стоимости пути. Это позволит алгоритму эффективно находить оптимальный путь в графе с учетом стоимости переходов и ограничений.

Разработка графического интерфейса пользователя, который позволит наглядно визуализировать процесс работы алгоритма A*. Графический интерфейс будет отображать граф, вершины, ребра и текущий путь, найденный алгоритмом.

Применение алгоритма A*:

Алгоритм A* находит широкое применение в области искусственного интеллекта, компьютерных игр, робототехники и планирования маршрутов. Он позволяет находить оптимальный путь с учетом сложностей и ограничений, что является важным аспектом во многих задачах. Реализация алгоритма A* на языке Java и его визуализация позволит лучше понять его работу и применение в реальных проектах, а также развить навыки программирования на языке Java.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Общие исходные требования

a. Приложение должно иметь графический интерфейс

b. Интерфейс должен быть легким для понимания и использования

1.2 Требования к вводу исходных данных

a. Входные данные задаются в правой части приложения

b. Входные данные задаются в виде сеточного поля с клетками

c. Каждая клетка может быть стартовой, конечной, проходимой и не проходимой

1.3 Требования к визуализации

a. Представление графа

- Граф будет представлен в виде сеточного поля
- Каждая клетка поля может быть 2-х типов (помимо стартовой и конечной клетки)
    - Свободная
    - Препятствие

b. Элементы управления

- Установка размера поля
- Выбор типа для каждой клетки
- Возможность выбрать эвристическую оценку из следующего списка:
- Евклидово расстояние
- Манхэттенское расстояние
- Диагональное расстояние
- Запуск алгоритма в пошаговом/непрерывном режиме
- Сброс поля
- Сохранение текущего поля в txt файл
- Загрузка поля из txt файла

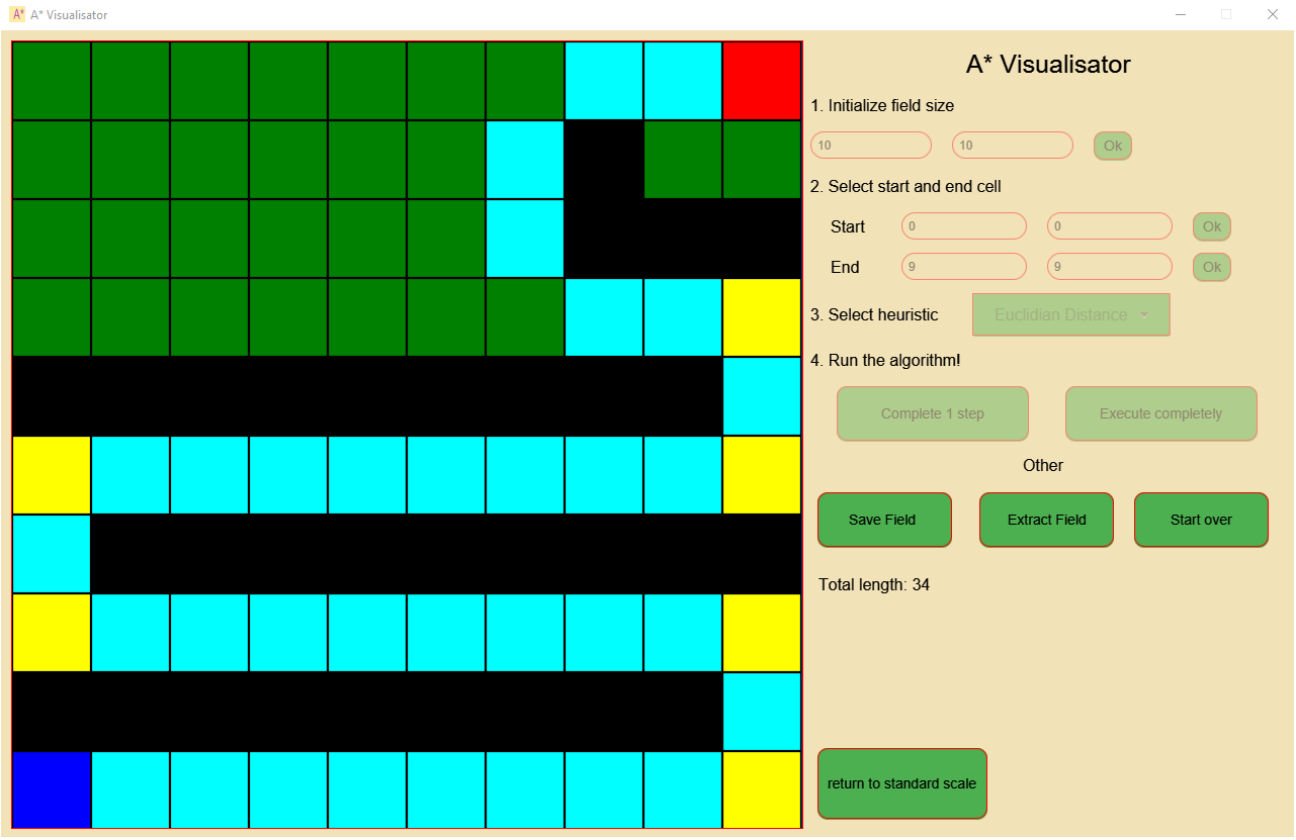

Рисунок 1 – Макет приложения

1.4 Формат выходных данных

По завершении работы, приложение демонстрирует:

- получившийся кратчайший путь между двумя выбранными вершинами графа
- длину пути

1.5 Архитектура программы.

Для реализации программы была выбрана структура Model-View-Controller. То есть вся реализация будет разделена на три компонента: контроллер, модель, графика. Схема данной архитектуры представлена на рис. 2.

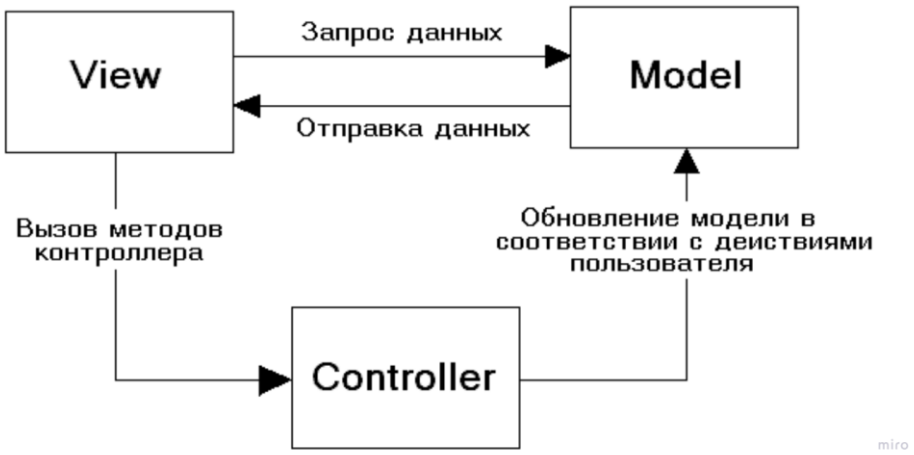

Рисунок 2 — Схема MVC

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

1. Разработка прототипа до 5-7 июля

- Создание макета графического интерфейса (компонента View)

- Реализация некоторых классов, классов-обработчиков событий в компоненте Controller (ввод/вывод, инициализация начальных данных)

- Реализация методов/классов в компоненте Model, которые будут использоваться для инициализации начальных данных.

2. Разработка 1-ой версии до 7-10 июля

- Реализация алгоритма A*

- Обеспечение взаимодействия с пользователем с помощью графического интерфейса, вывод результата

- Добавление классов-исключений

- Исправление замечаний

3. Разработка 2-ой версии до 10-12 июля

- Добавление тестирования

- Написание отчета

- Исправление замечаний

4. Сдача финальной версии 12-13 июля

- Исправление замечаний

2.2. Распределение ролей в бригаде

Сочков И. С.: реализация Model, Controller в паттерне MVC, написание отчета

Овчинников М.Ю.: реализация View, Controller в паттерне MVC, тестирование, создание UML-диаграммы

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Каждая клетка поля, которую можно наблюдать на GUI, описывается классом Cell. Данная структура содержит данные о координатах, типе клетки, информацию, о наличии соседей данной клетки и последующие методы, определяющие и меняющие данные.

Также в коде содержатся перечисления, в которых располагается информация о типах клетки (CellType), видах эвристической оценки (Heuruistics).

3.2. Реализация алгоритма

Одним из основных методов кода естественно является метод A star.

Алгоритм A* использует эвристическую функцию для оценки стоимости прохода через каждую клетку. В коде используются различные переменные и структуры данных: openSet (приоритетная очередь (PriorityQueue), содержащая клетки, которые еще не были полностью исследованы); closeSet (множество HashSet, содержащее уже исследованные клетки); currentCell (текущая обрабатываемая клетка); endCell (целевая клетка, до которой нужно найти кратчайший путь); neighbors (список соседних клеток для текущей клетки); cameFrom (карта HashMap, содержащая информацию о том, из какой клетки пришли в каждую клетку); tentativeGScore (временное значение g-стоимости (стоимости пути от начальной клетки до текущей клетки) для соседних клеток);

В данном алгоритме участвуют следующие методы: delBlockedCellFromQueue (метод, который удаляет заблокированные клетки из очереди openSet перед запуском алгоритма); calculateHeuristic (метод, вычисляющий эвристическую оценку (h-стоимость) для соседней клетки).

Действия алгоритма. Сначала алгоритм удаляет заблокированные клетки из очереди openSet. Если openSet не пустая, извлекает клетку с наименьшей оценкой f = g + h из openSet. Затем добавляет текущую клетку в closeSet. Если текущая клетка равна целевой клетке (endCell), возвращает координаты этой клетки.

После этого получается список соседних клеток для текущей клетки. Для каждой соседней клетки выполняет следующие действия. Если соседняя клетка не заблокирована, то вычисляется временное значение g-стоимости для соседней клетки. Если же временное значение g-стоимости меньше текущей g-стоимости соседней клетки, то обновляется информация о предыдущей клетке (cameFrom) для соседней клетки.

Алгоритм устанавливает новое значение g-стоимости и h-стоимости для соседней клетки. Затем добавляет соседнюю клетку в openSet. Если в очереди openSet есть элементы, возвращает координаты следующей клетки в очереди (клетки с наименьшей оценкой f). Если же очередь openSet пуста, возвращает null.

Этот код представляет одну итерацию алгоритма A* и может быть вызван повторно до тех пор, пока не будет достигнута целевая клетка или пока все доступные пути не будут исследованы.

3.3 Графический интерфейс

Графический интерфейс реализован с помощью библиотеки JavaFX. В начале программа загружает макет формата fxml, затем функционал дополняется контроллером, содержащим обработчики событий. Также имеется файл .css, который работает над стилями объектов окна. Рабочее окно можно логически разделить на две части: панель инструментов и поле.

Панель инструментов включает в себя возможность выбрать действие над полем (создание поля, выбор начальной/конечной клетки и т.д.). При этом некоторые клавиши будут блокироваться, например, клавиши инициализации поля недоступны после отработки алгоритма. Также есть возможность приближать поле, сохранять его и загружать. Каждая кнопка может выбрасывать исключения, это сделано с той целью, чтобы предупредить пользователя о его некорректных действиях, например, ввод неверных данных, попытка запустить алгоритм без инициализации, загрузить неверное поле и т.д.

Поле включает в себя массив клеток CellView, которые наследуются от класса javafx.shape.Rectangle. Наследование позволило задавать цвета клеток не

на прямую, а согласно их типу. Рисование клеток происходит в контейнере Pane, который в свою очередь лежит в контейнере ScrollPane, чтобы поле можно было масштабировать. Это сделано для случая, когда поле слишком большое. Также была реализована кнопка, которая позволяет установить стандартный масштаб.

3.4 Controller

Класс-контроллер нужен для связи между моделью и графическим представлением. Класс контроллер реализуется следующим образом: он хранит в себе классы модели и представления, а также все возможные кнопки доступные в графическом интерфейсе. Для каждой кнопки устанавливается обработчик события, который прослушивает нажатия на кнопки, когда определенная кнопка нажимается, происходит срабатывание соответствующего обработчика, тот в свою очередь выполняет действия в модели и в графическом представлении, а также обрабатывает возможные исключения. Таким образом происходит общение между моделью и графическим интерфейсом.

UML-диаграмма классов представлена в приложении А.

4. ТЕСТИРОВАНИЕ

4.1 Тестирование обработки исключительных ситуаций и интерфейса

Было проведено ручное тестирование, рассмотрены исключительные ситуации и реакция программы на них:

1) Ввод некорректных данных

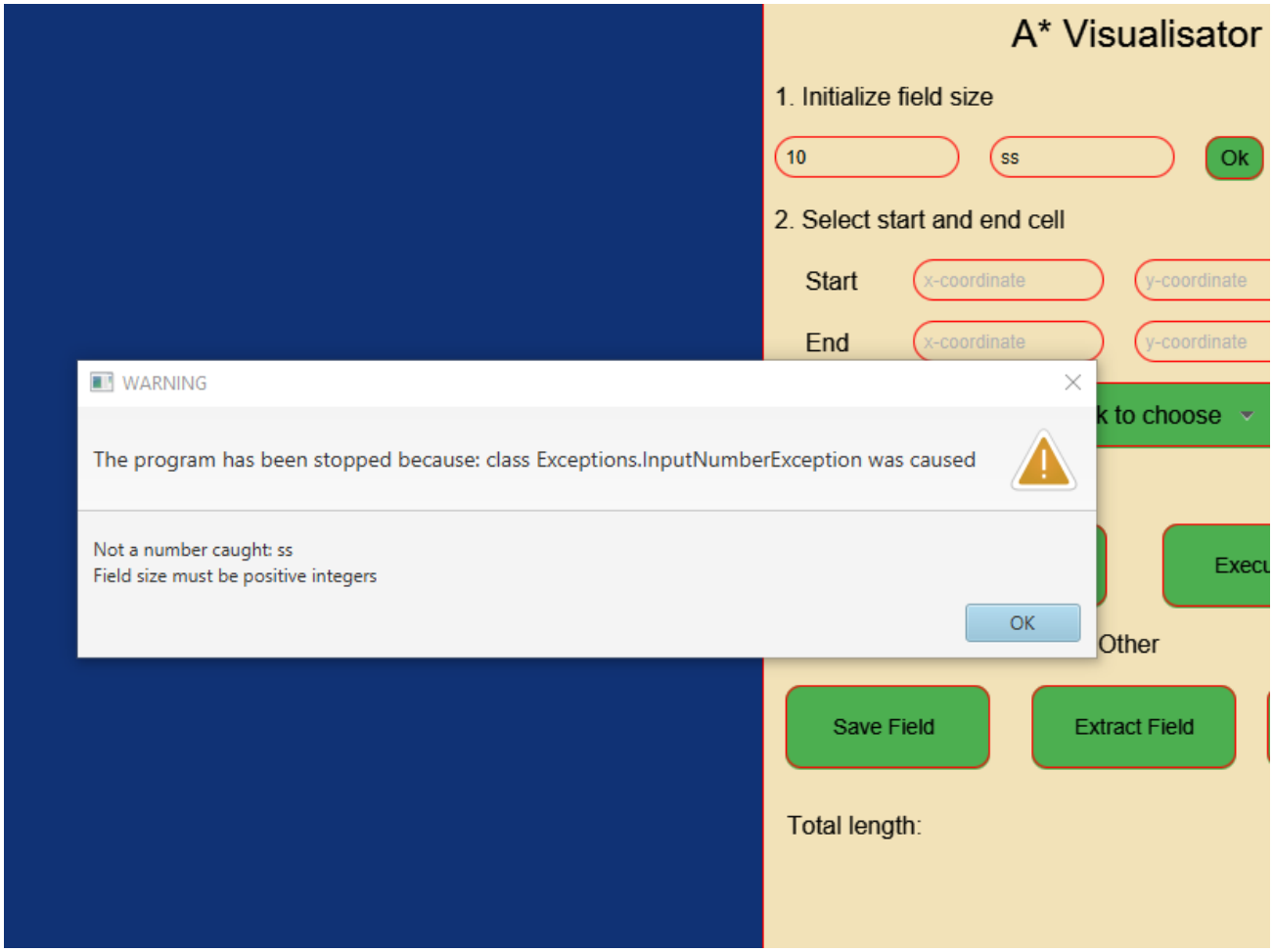

Рисунок 3 – попытка ввести некорректные данные

Если ввести в поле что то, кроме целого числа, то будет выброшено исключение,причем программа не завершит работу,а будет ожидать ввод. Такая же логика и для задания стартовой и конечной точки.

2) Попытка запустить алгоритм без инициализации поля

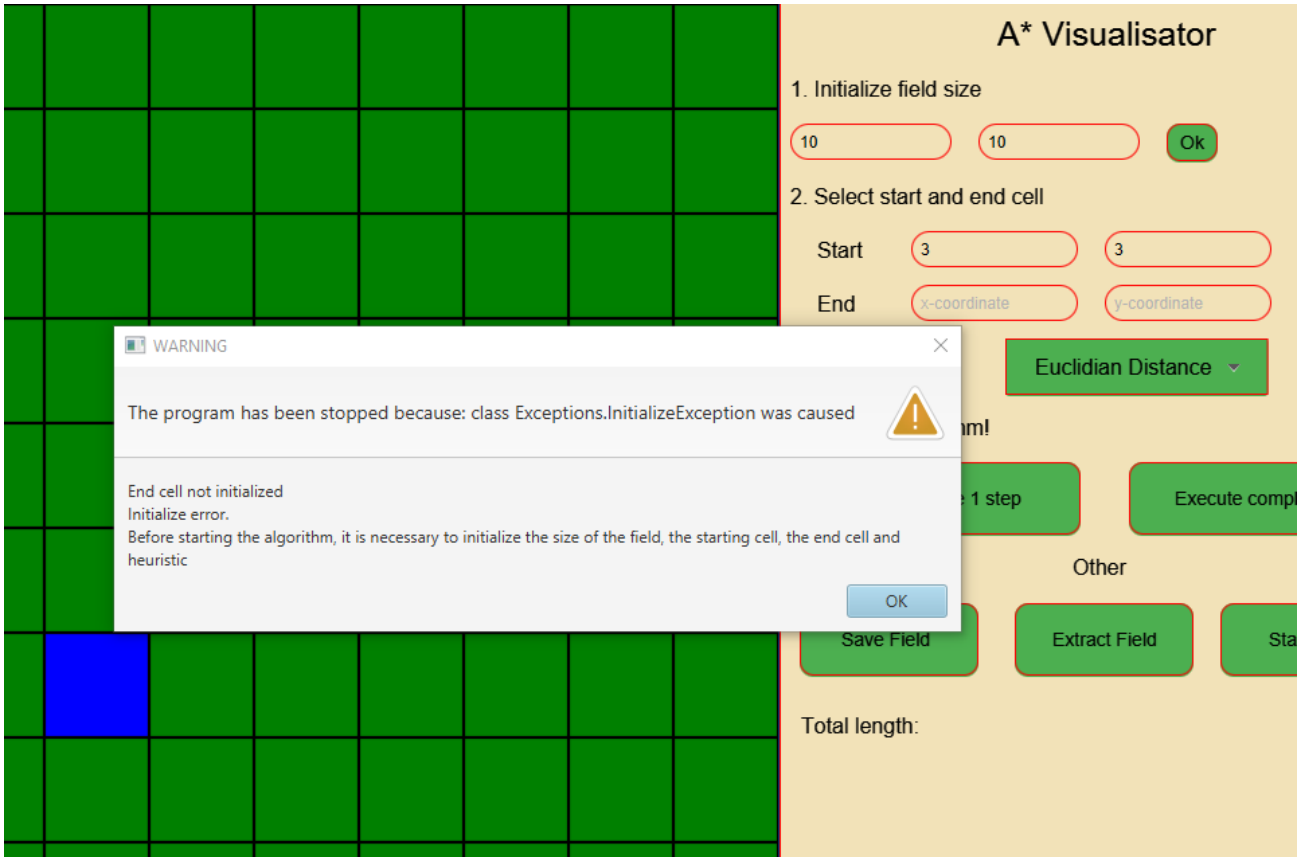

Рисунок 4 – попытка запустить алгоритм без полной инициализации

Если попробовать запустить алгоритм без инициализации, то программа сообщит об этом, напишет, что именно не инициализированно и что нужно инициализировать в общем случае.

3) Попытка задания конечной/стартовой клетки за границами поля

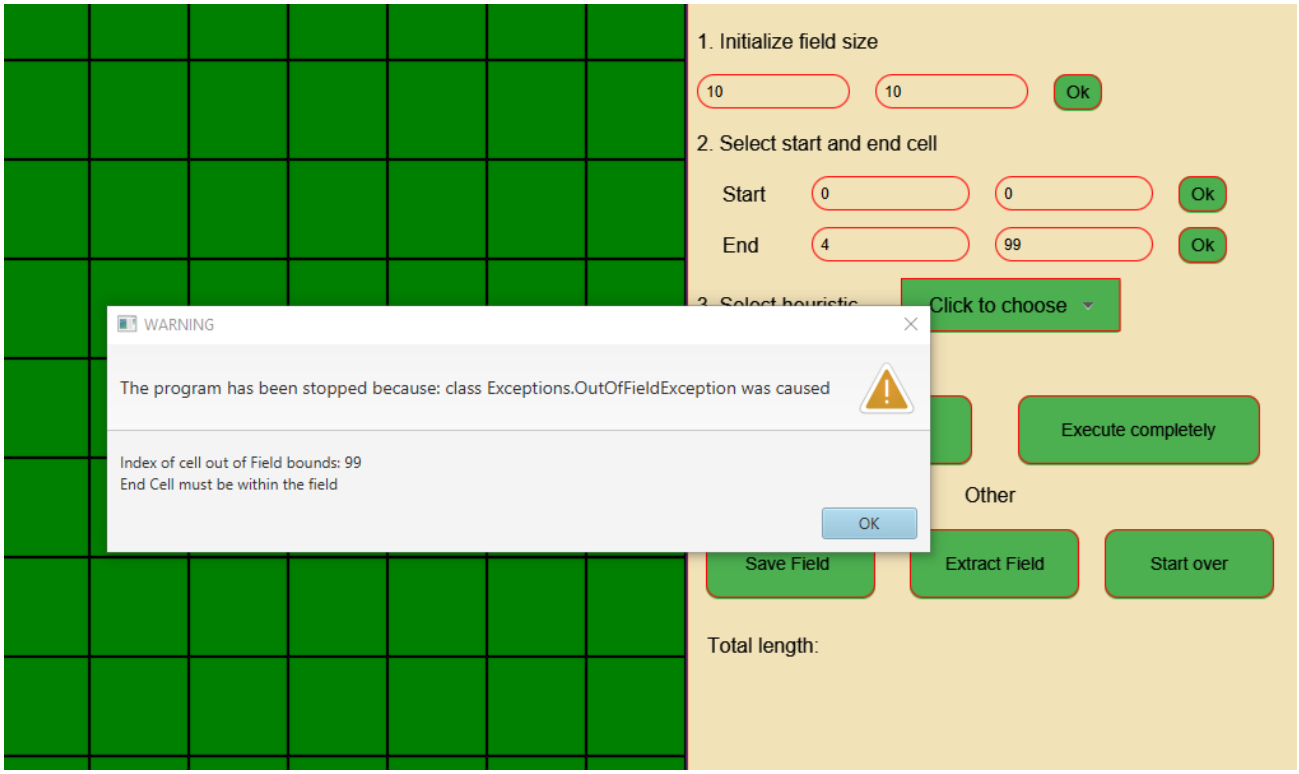

Рисунок 5 – попытка задания конечной клетки за границами поля

Программа верно обрабатывает эту ошибку и указывает в чем проблема.

4) Попытка сохранить пустое поле в файл

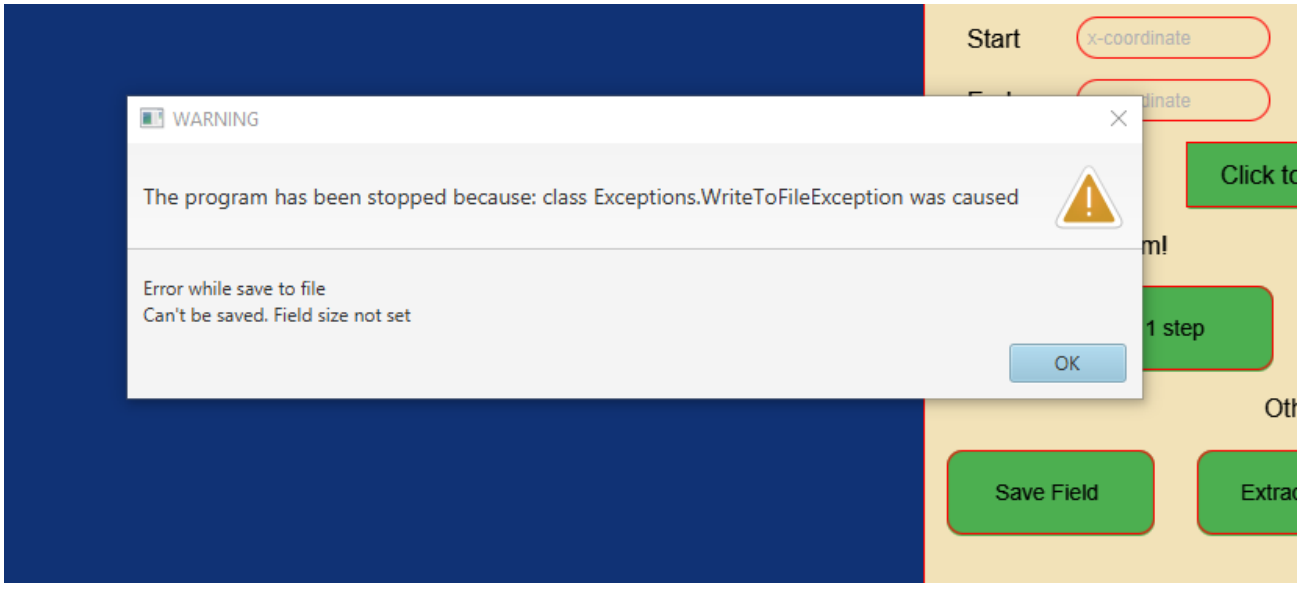

Рисунок 6 – попытка сохранить пустое поле

Программа не позволяет сохранить пустое поле, для сохранения как минимум требуется создать поле какого-то размера, а так-же указать эвристику.

5) Попытка загрузить неверное поле

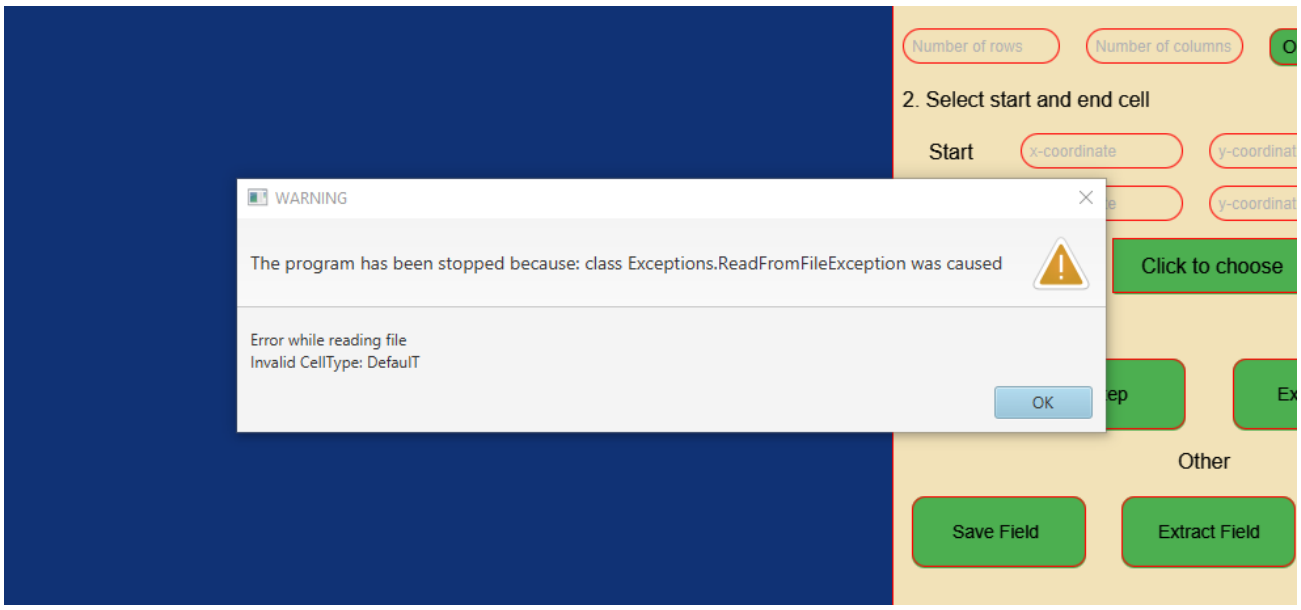

Рисунок 7 – попытка задать неверный тип клетки в файле

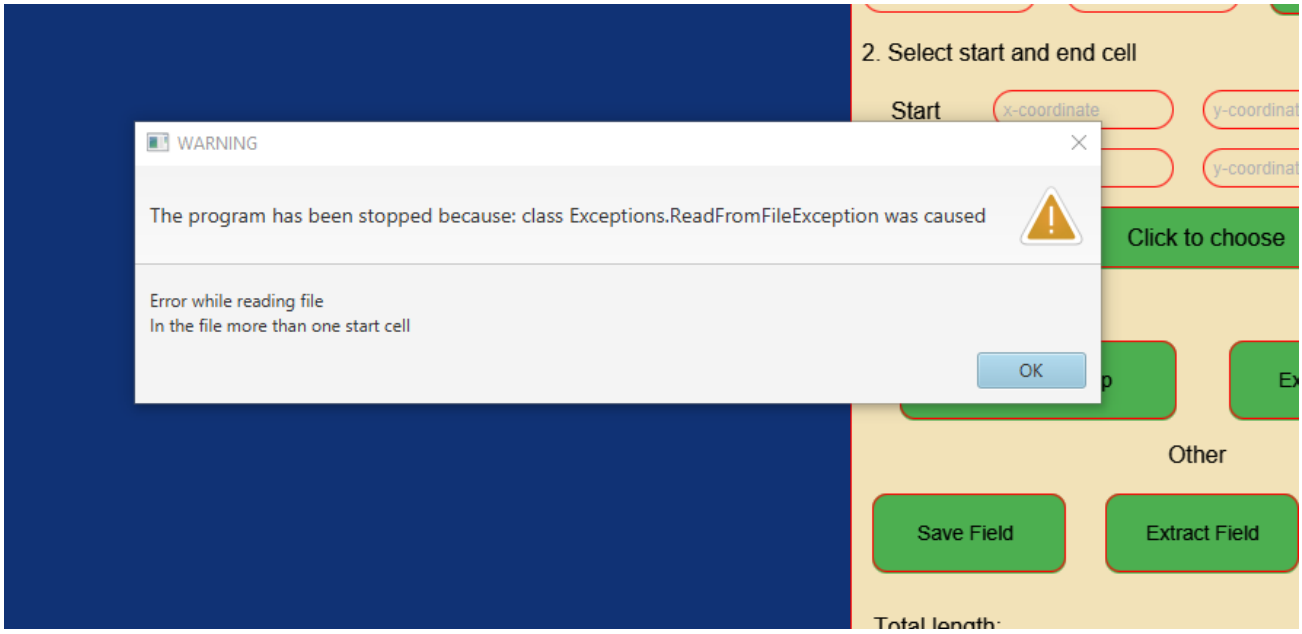

Рисунок 8 – попытка задать две стартовые клетки

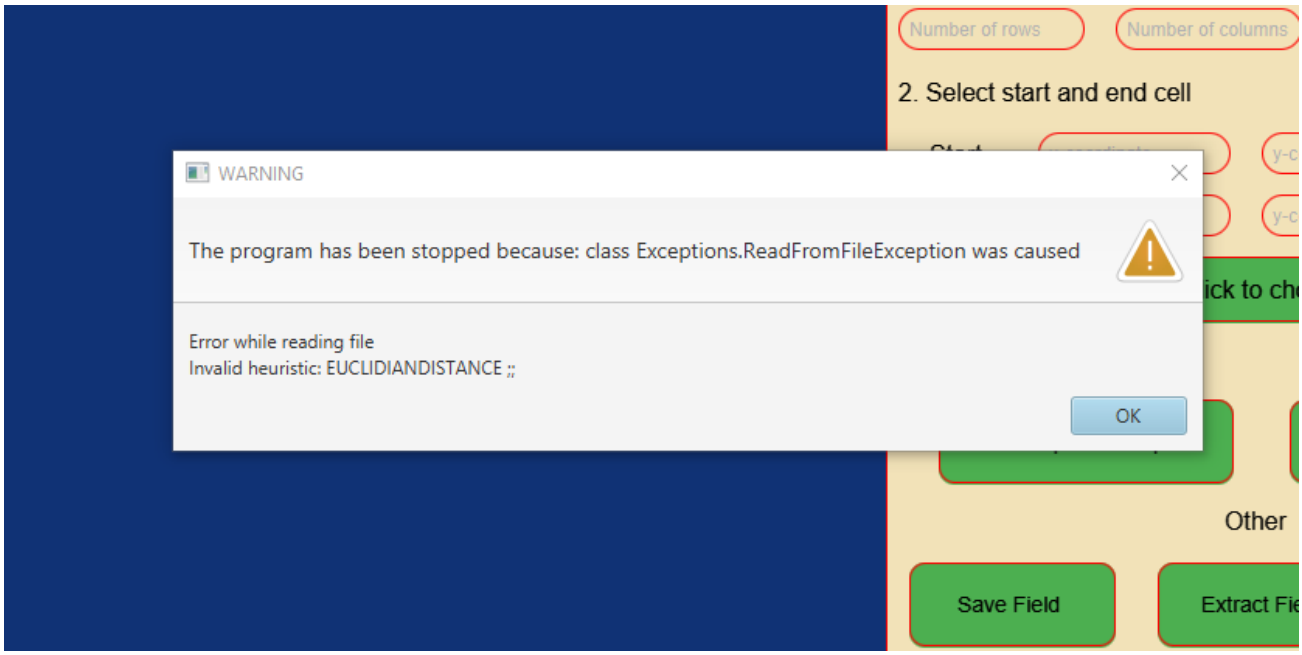

Рисунок 9 – попытка задать неверный тип эвристики

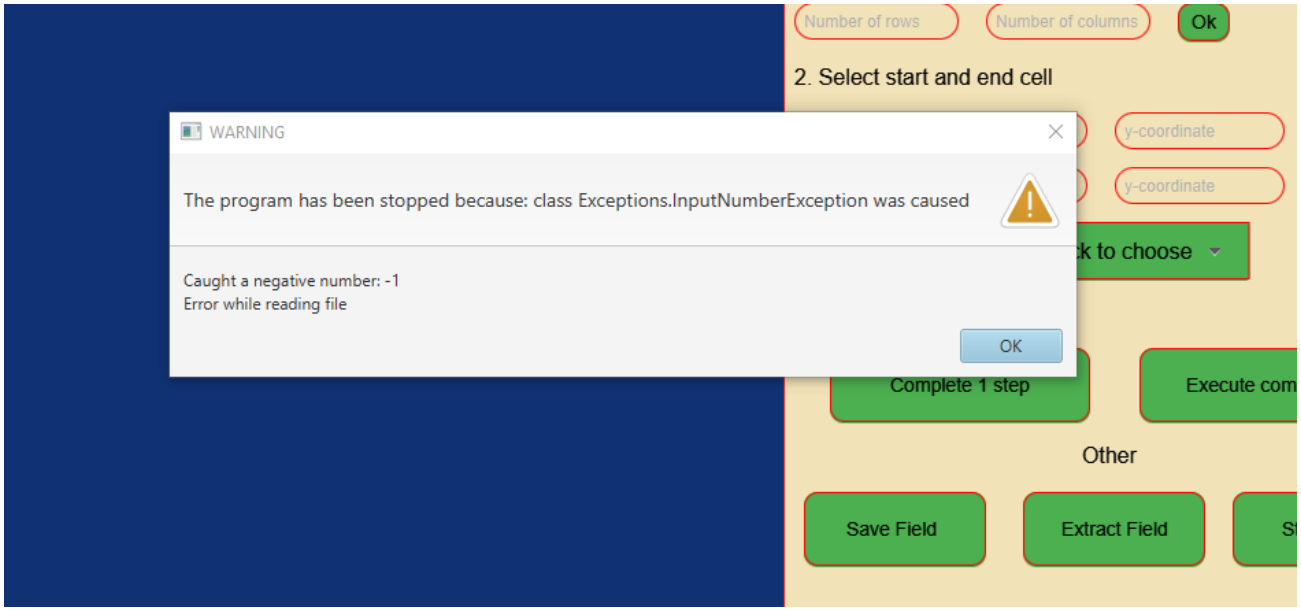

Рисунок 10 – попытка задать неверный тип клетки в файле

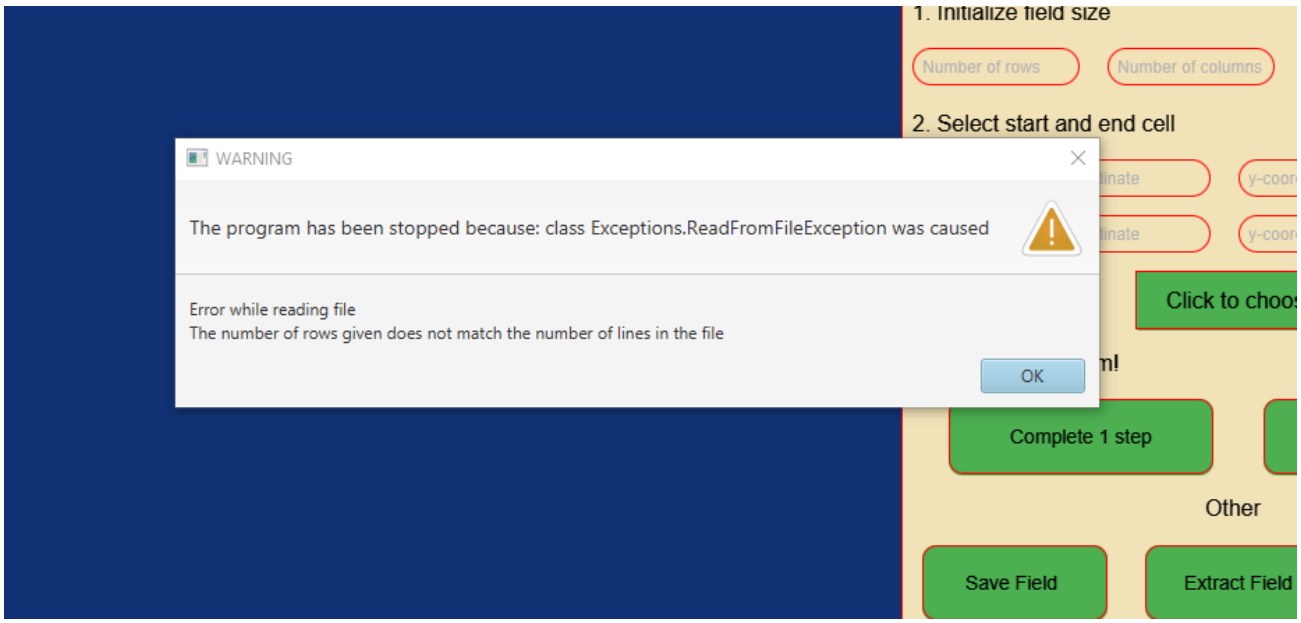

Рисунок 11 – размер поля заданный в файле не совпадает с размером переданного поля

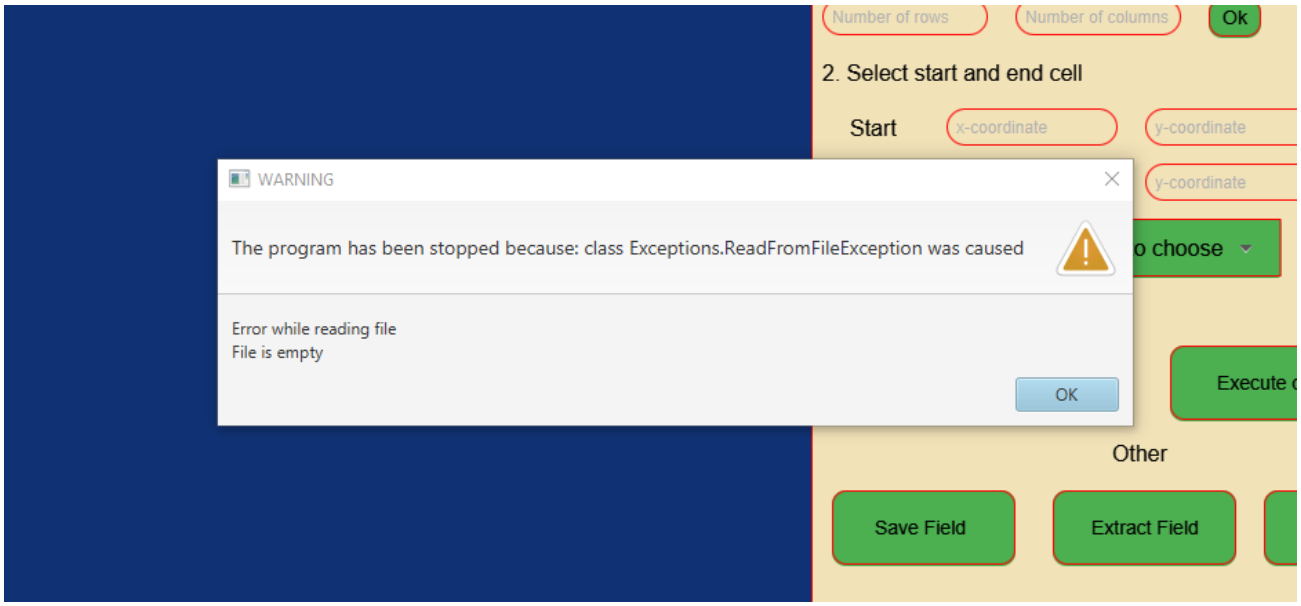

Рисунок 12 – попытка считать пустое поле

4.2 Тестирование алгоритма

1) Полу-лабиринт

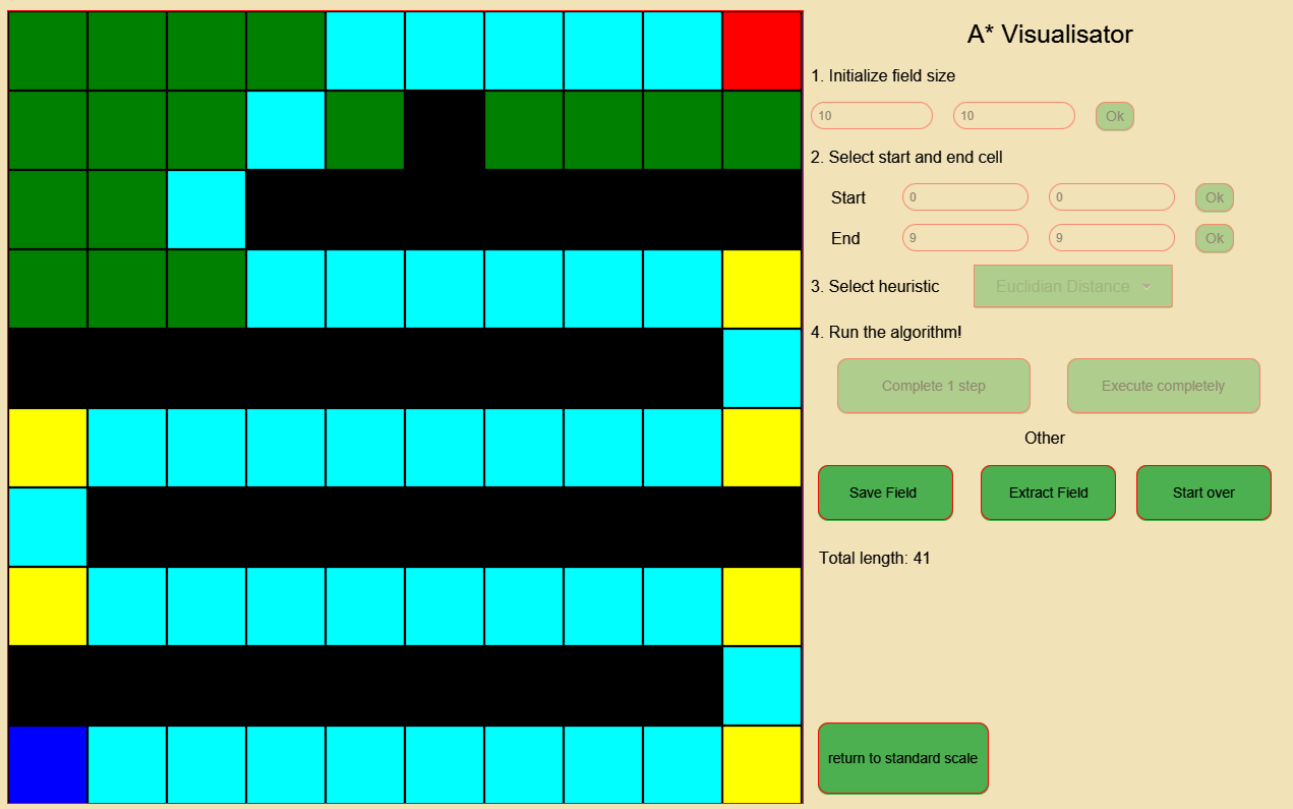

Рисунок 13 – результат работы алгоритма для полу-лабиринта

2) Недостижимая финальная клетка

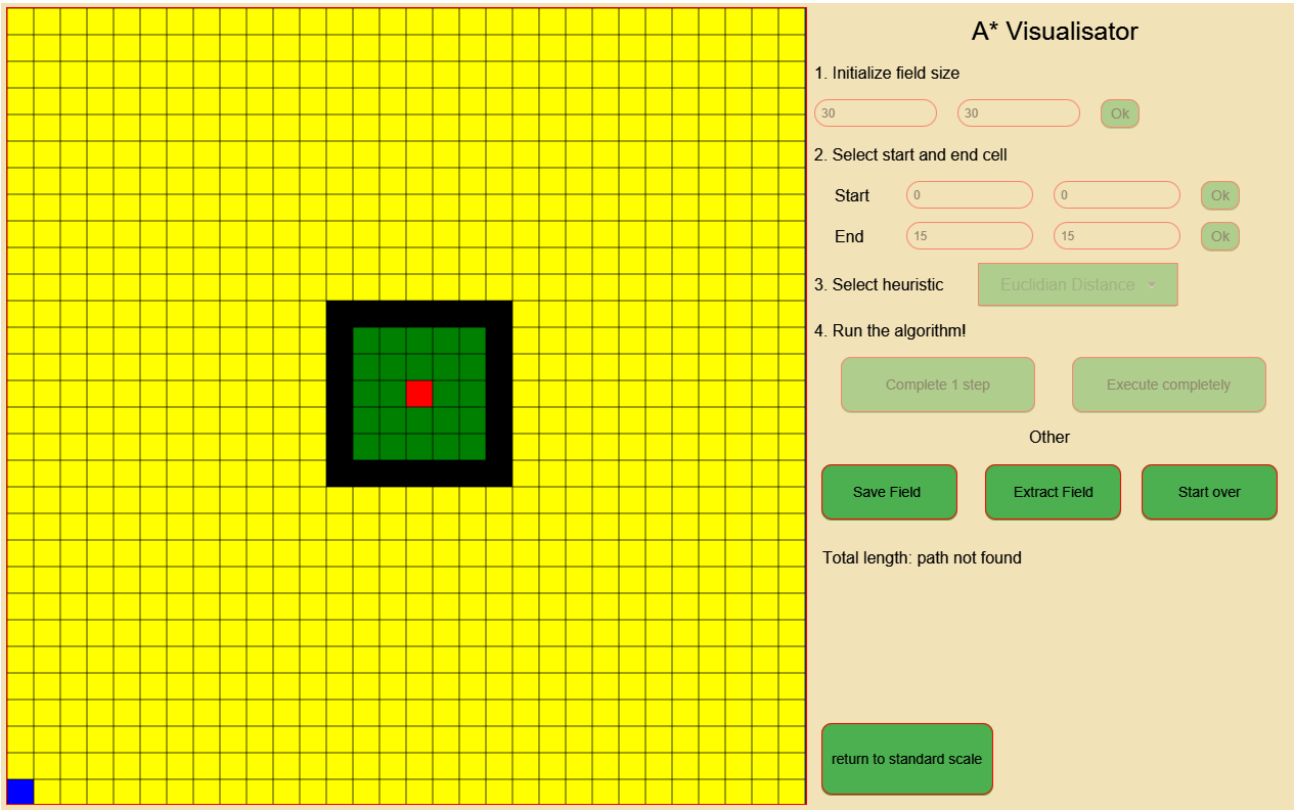

Рисунок 14 – результат работы алгоритма при условии недостижимости финальной клетки

3) Прямой путь без препядствий

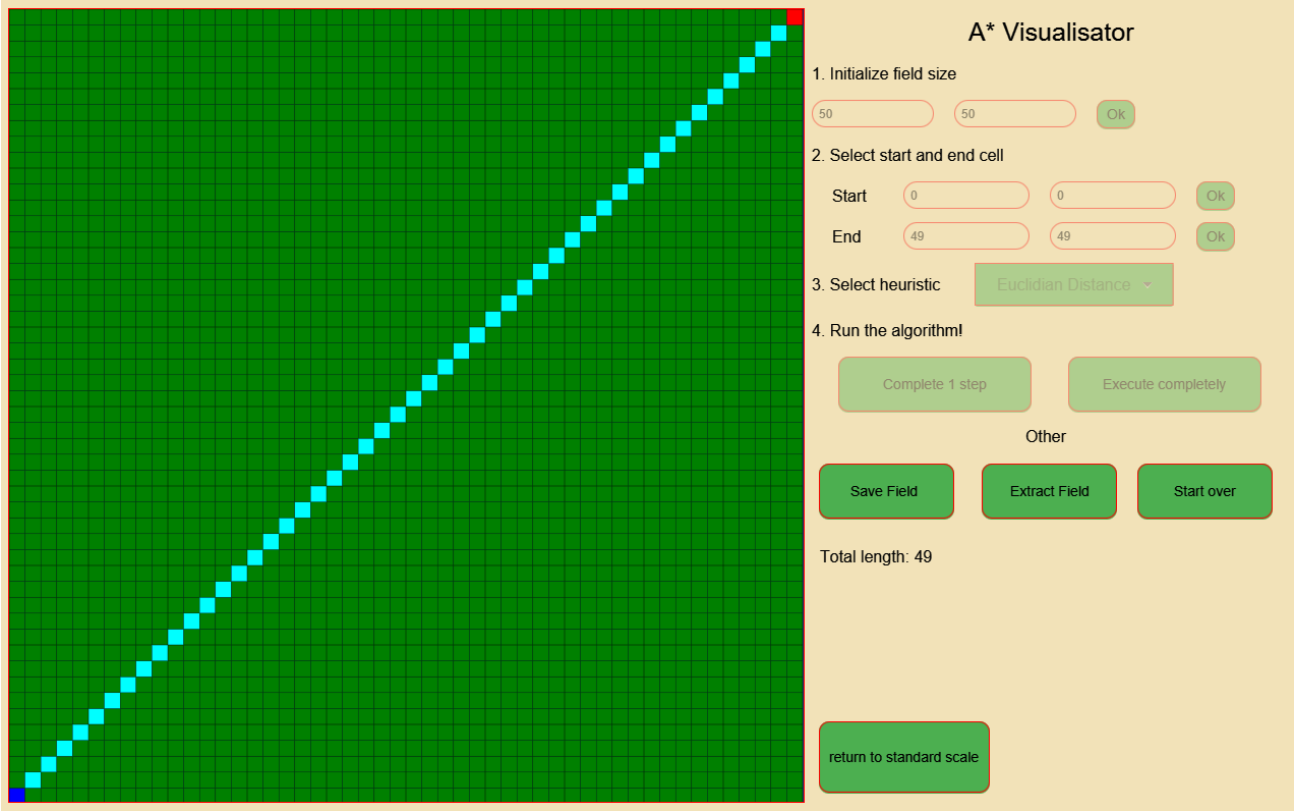

Рисунок 15 – результат работы алгоритма для прямого пути без препядствий

4) Поле размером 2x2

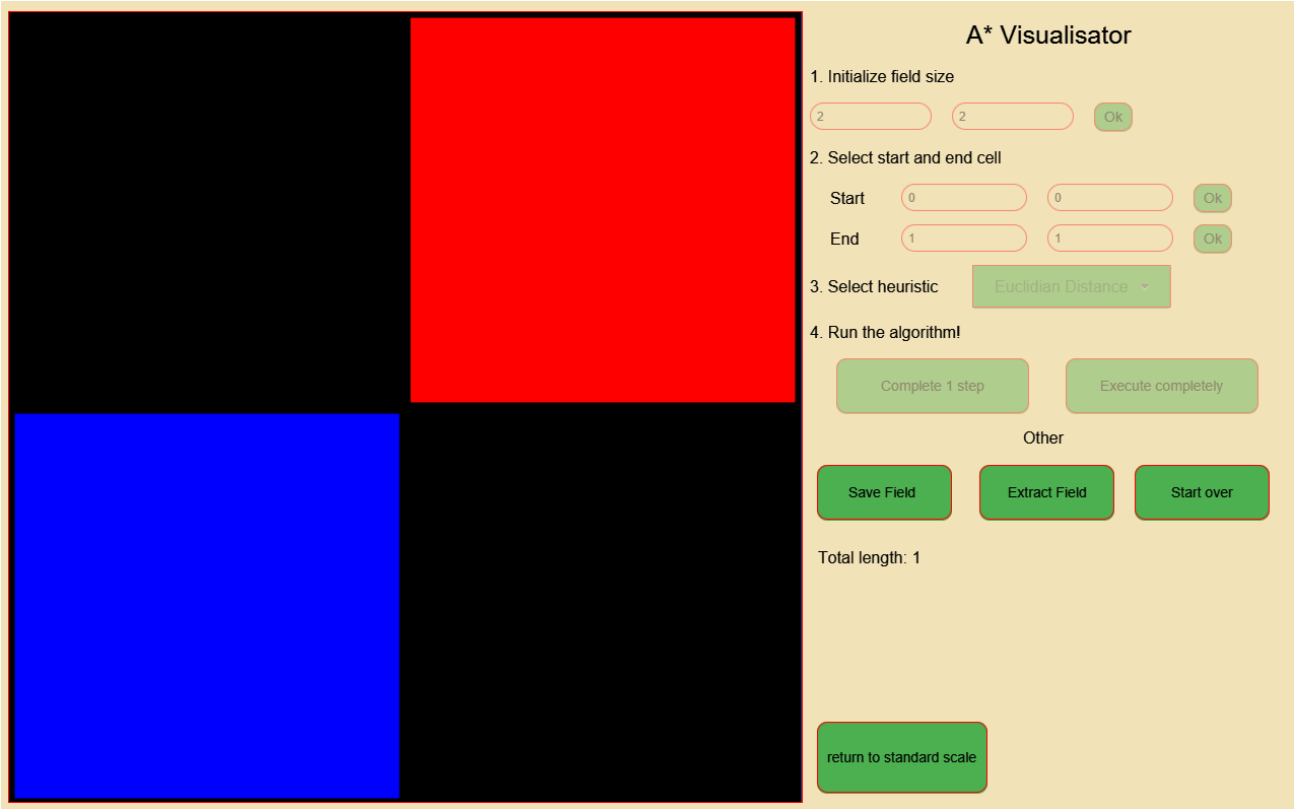

Рисунок 16 – результат работы алгоритма для начальной и конечной клетки, находящихся на расстоянии 1 друг от друга

5) Поле размером 100x100 с препядствиями

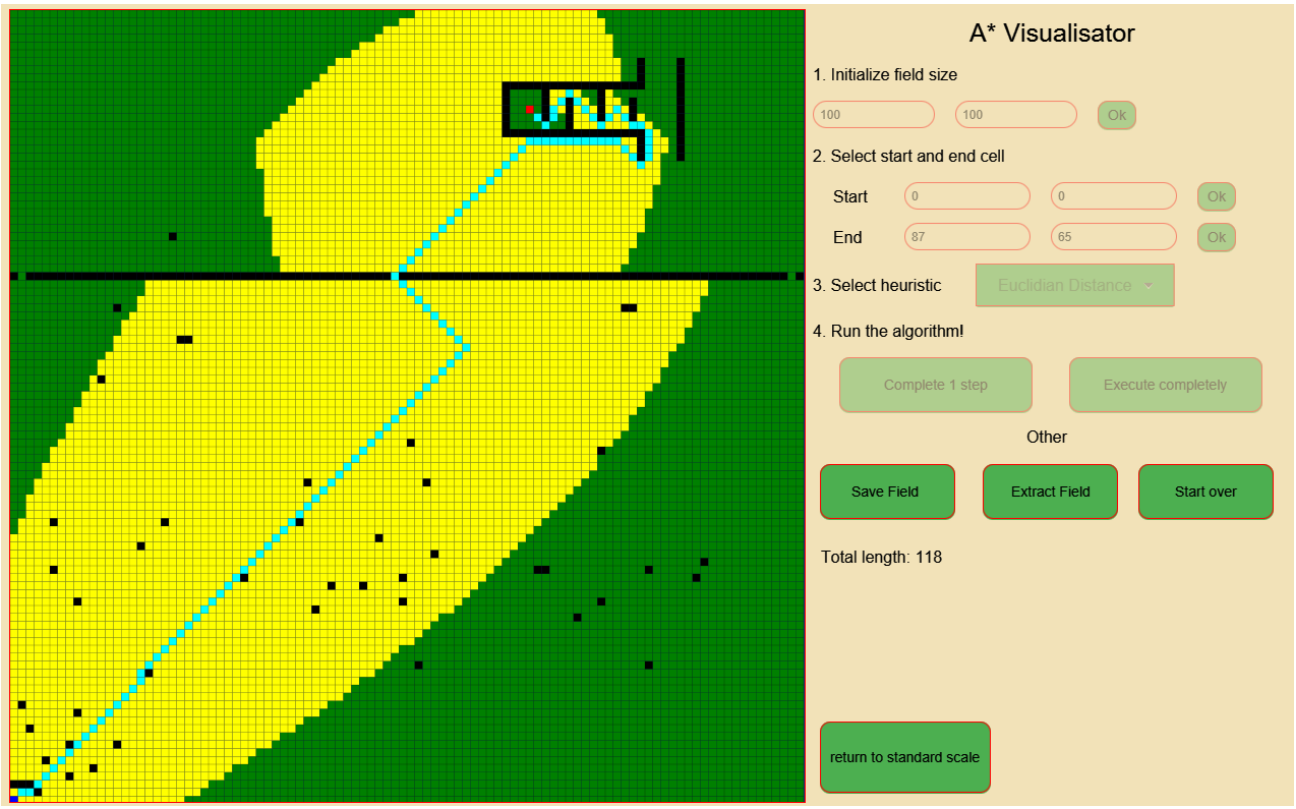

Рисунок 17 – результат работы алгоритма для поля размером 100x100

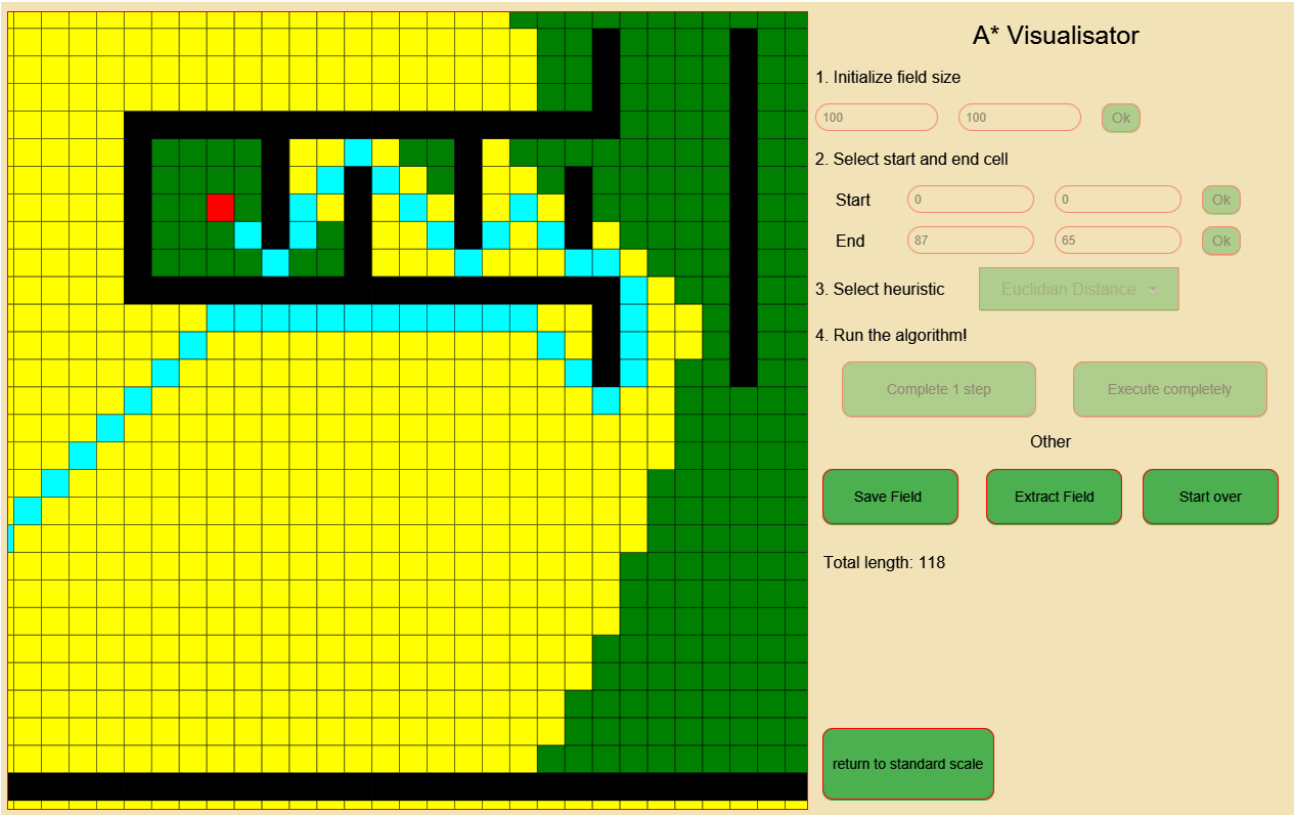

Рисунок 18 – результат работы алгоритма в приближении

6) Лабиринт

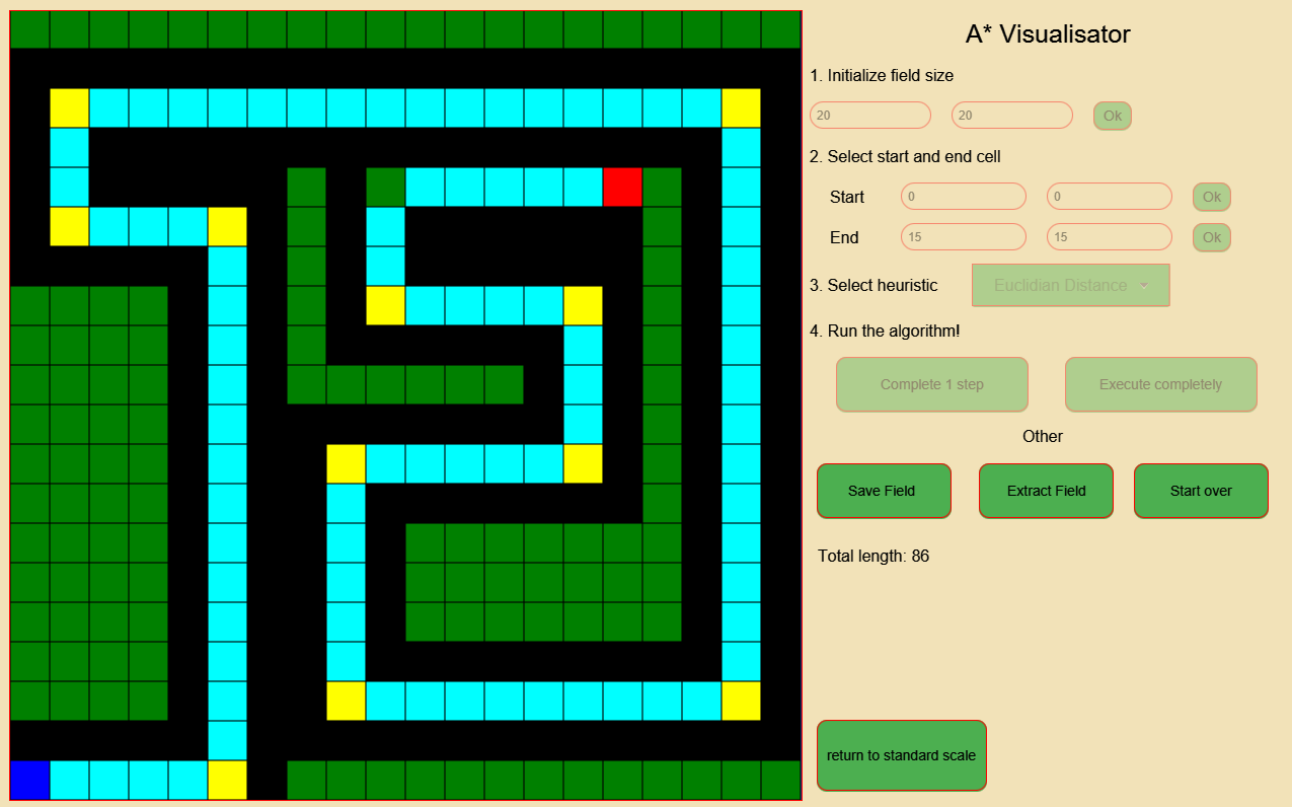

Рисунок 19 – результат работы алгоритма для лабиринта

7) Поле состоящее только из начальной, конечной и закрашенных клеток

Рисунок 20 – результат работы алгоритма для полностью непроходимого поля

ЗАКЛЮЧЕНИЕ

Изучение Java и реализация алгоритма A* с визуализацией представляют увлекательный и практически значимый путь в области программирования. В ходе изучения Java мы освоили один из наиболее популярных и мощных языков программирования, который широко применяется в различных областях, включая разработку мобильных приложений, веб-приложений, игр и многие другие.

Алгоритм A* является одним из наиболее эффективных алгоритмов поиска кратчайшего пути и широко применяется в различных задачах, таких как планирование маршрутов, искусственный интеллект, игровое программирование и робототехника. Его эффективность основана на комбинации оценки стоимости прохода через каждую клетку (эвристики) и динамического выбора наилучшего пути на основе этих оценок.

Реализация алгоритма A* с визуализацией позволяет наглядно представить работу алгоритма, отображая промежуточные шаги поиска пути на графическом интерфейсе. Это важно не только для понимания самого алгоритма, но и для визуального анализа и отладки, а также для обучения и общения с другими разработчиками.

В процессе реализации мы приобрели не только практические навыки программирования на Java, но и глубже поняли принципы работы алгоритма A*. Мы научились работать с различными структурами данных, такими как приоритетные очереди и хэш-карты, а также использовать объектно-ориентированный подход для управления клетками и их свойствами.

Кроме того, реализация визуализации позволила нам развить навыки работы с графическим интерфейсом и использовать графические элементы для отображения данных и взаимодействия с пользователем.

Изучение Java и реализация алгоритма A* с визуализацией позволили нам расширить наши знания и навыки в программировании, а также применить их на практике в реализации полезного и интересного алгоритма. Этот опыт может стать прочной основой для дальнейшего развития в области разработки

программного обеспечения и решения сложных задач, требующих поиска оптимальных путей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Алгоритм A*: https://neerc.ifmo.ru/wiki/index.php?title=%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_A*
2. JavaDoc JavaFX: https://openjfx.io/javadoc/20/
3. Репозиторий бригады на github https://github.com/Maxim2121512/Practice/tree/finalVersion_v2

ПРИЛОЖЕНИЕ А

UML-ДИАГРАММА