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

ОТЧЕТ

по учебной практике

Тема: Реализация алгоритма A* на языке Kotlin с визуализацией.

Студентка гр. 1303 _____ Королева П.А

Студент гр. 1303 _____ Гирман А.В

Студент гр. 1303 _____ Самохин К.А

Руководитель _____ Шестопалов Р.П.

Санкт-Петербург
2023

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студентка Королева П.А группы 1303

Студент Гирман А.В группы 1303

Студент Самохин К.А группы 1303

Тема практики: Алгоритм A*: Нахождение кратчайшего пути в графе

Задание на практику:

Командная итеративная разработка визуализатора алгоритма на Kotlin с графическим интерфейсом.

Алгоритм: A*

Сроки прохождения практики: 30.06.2023 – 13.07.2023

Дата сдачи отчета: 12.07.2023

Дата защиты отчета: 12.07.2023

Студентка		Королева П.А
Студент		Гирман.А.В
Студент		Самохин К.А
Руководитель		Шестопалов Р.П

АННОТАЦИЯ

Целью проекта является получение навыков программирования на Kotlin и создание программы, визуализирующей работу алгоритма A*, поиска кратчайшего пути во взвешенном графе.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Главной целью работы было реализовать алгоритм A* для поиска кратчайших путей на карте и представить его в виде приложения с графическим интерфейсом.

Для корректной работы алгоритма реализуем очередь с приоритетом, в которой будут храниться клетки-кандидаты для перехода.

В реализованную очередь с приоритетом добавляем стартовую вершину. До тех пор, пока очередь не пуста, достаем из нее вершину с наименьшим значением эвристической функции и рассчитываем аналогичное значение для смежных вершин. Если очередная вершина ещё не была посещена, или существующая оценка больше только что вычисленной, значение для данной вершины обновляется. После этого вершина и её приоритет помещаются в очередь. Если достигнута конечная вершина, поиск прекращается.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1 – требования к вводу исходных данных

В начале работы, программа в диалоговом окне запрашивает размеры карты. Создается прямоугольная область с заданными размерами, все ячейки по умолчанию имеют тип «трава». Тип можно поменять щелчком мыши по клетке. На рисунке 1 изображено поле при запуске программы и способ задания типа клеток .

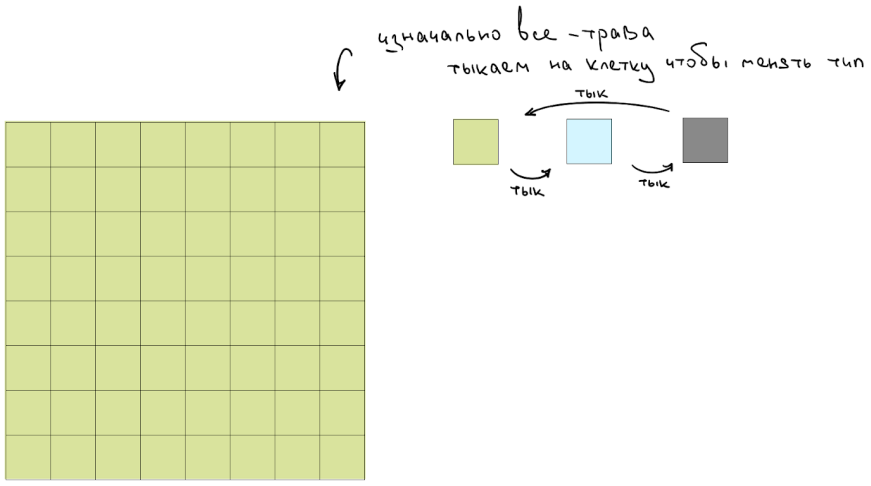

Рисунок 1 – Задание карты

Чтобы отметить стартовую и финишную точки, нужно использовать кнопки справа.

Также должна быть предусмотрена возможность задания карты через файл.

1.1.2 – требования к визуализации

Окно разделено на две части:

В левой демонстрируется карта, на которой отмечается каждый шаг алгоритма: на каждой клетке, обработанной алгоритмом, указываются:

g – расстояние от старта до клетки

h – эвристическая функция

f – приоритет.

На клетке есть стрелочка, указывающая на родительскую ячейку, т.е откуда алгоритм пришел в данную ячейку.

Клетка может находиться в трех состояниях, в соответствии с алгоритмом:

- *не на рассмотрении*, выглядит как обычная клетка
- *на рассмотрении* (т.е находится в открытом списке), немного серая
- *рассмотрена* (т.е находится в закрытом списке), темно серая

В правой части окна находится панель, на которой указаны для справки типы клеток, и три кнопки:

- *установить старт* – после нажатия на кнопку, можно нажать на любую проходимую ячейку чтобы в ней появился старт.
- *установить финиш* – аналогично, после нажатия на кнопку, можно нажать на ячейку чтобы она стала финишной.
- *построить путь* – запускает алгоритм A*

На рисунке 2 изображен макет приложения.

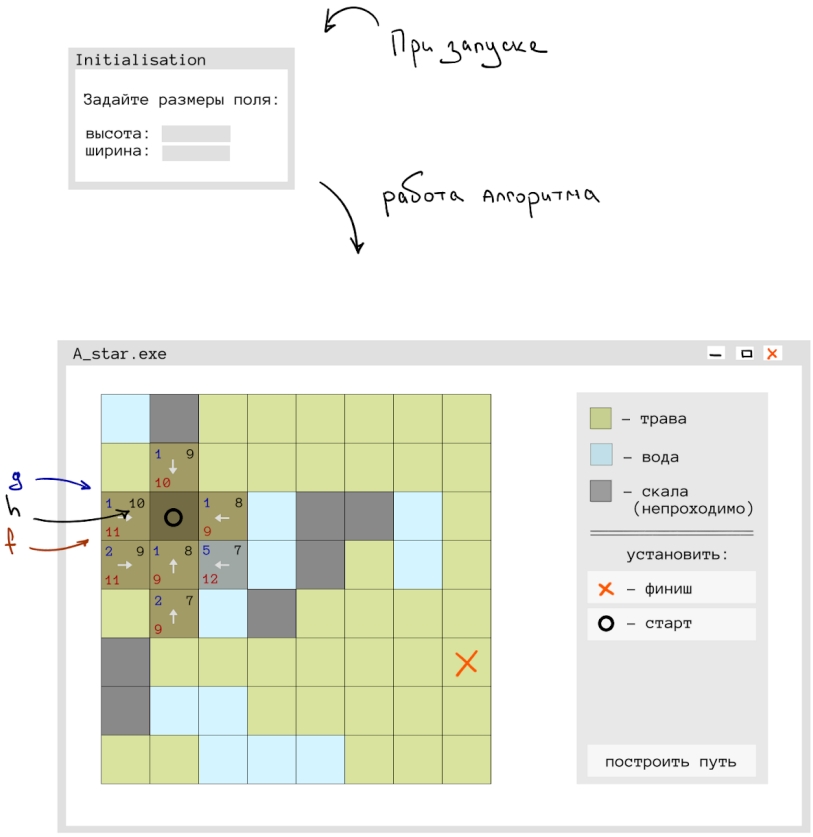

Рисунок 2 – Макет приложения

1.1.3 – Иерархия классов

На рисунке 3 представлена схема классов, логически ее можно поделить на три части: модель (Field, Cells), визуализация (FieldView, CellView) и контроллер (Controller), поддерживающий работу этих компонент, управляемый алгоритмом и пользователем.

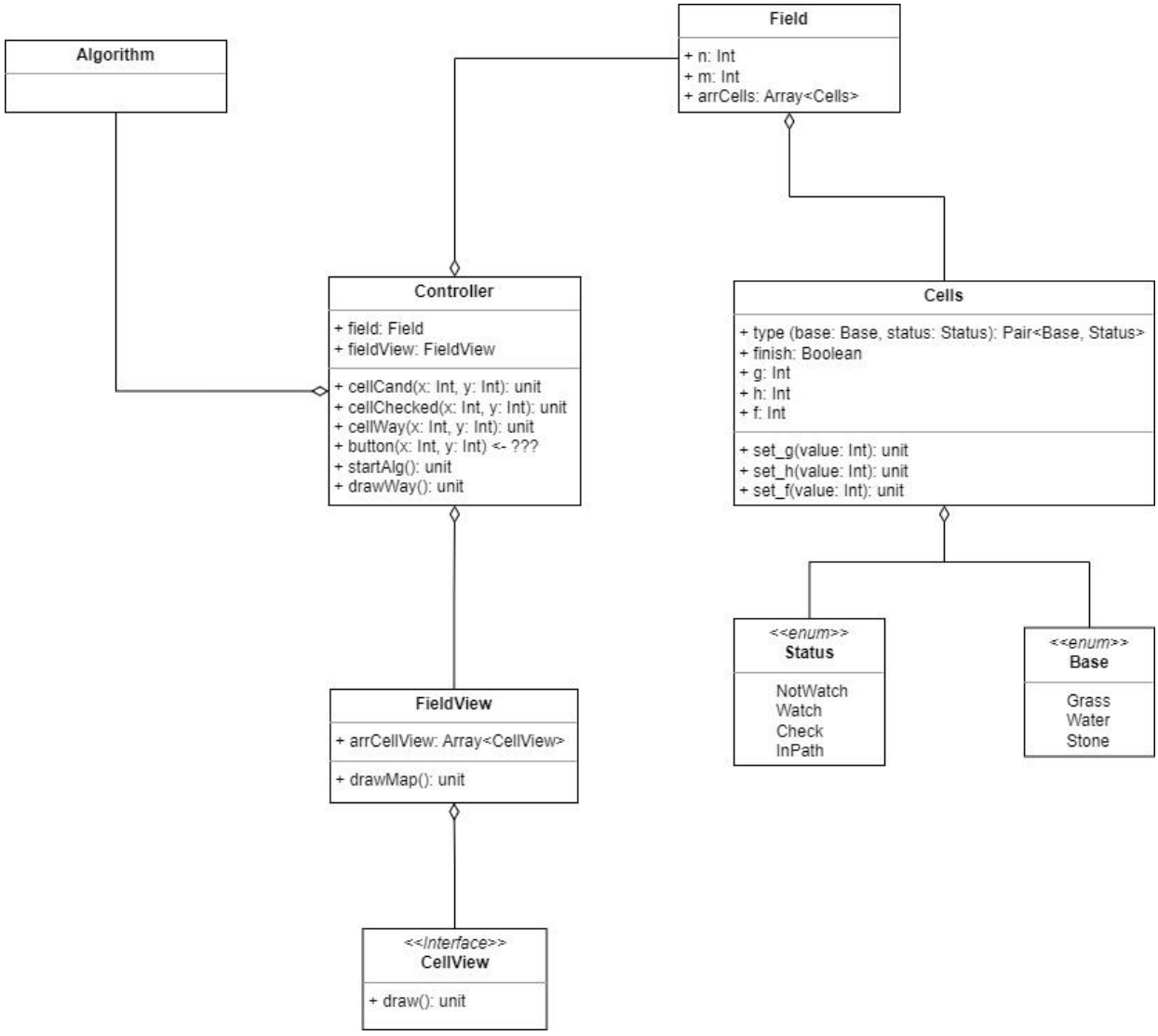

Рисунок 3 – Иерархия классов

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

Приблизительный план разработки приложения:

3 июля – Согласование спецификации и плана разработки

5 июля – Сдача прототипа: написание алгоритма, разработка диалогового окна, обработка нажатий.

7 июля – Сдача первой версии: окончательная разработка всех классов, визуализация пошагового выполнения алгоритма

10 июля – Сдача второй версии: исправление недочетов, полировка программы, обработка исключительных ситуаций.

12 июля – Финальная версия и отчет.

2.2. Распределение ролей в бригаде

Королева П. – отображение поля и клеток, интерфейс, реализация пошаговости алгоритма, отчет.

Гирман А. – диалоговые окна, кнопки, интерфейс, отлов ошибок.

Самохин К. – реализация алгоритма, классов клетки и поля, ввода из файла.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

Структуры данных:

3.1. Для алгоритма A*

Реализованы классы *Algorithm, Field, Cell, Heap.*

Algorithm реализует описываемый алгоритм.

Основные методы:

aStar() – метод используется для начальной инициализации алгоритма.

iteration() – метод выполняет извлечение очередной клетки из очереди и помечает её как посещённую. Если же отмеченная клетка является финишем, метод возвращает словарь переходов *roots.*

cellProcess() – метод рассматривает соседей последней извлечённой из очереди вершины. В случае, если на клетке установлен камень, или она находится за границами поля, она пропускается. Если же вершина ранее не была добавлена в очередь, или записанное значение приоритета меньше только что вычисленного, в очередь помещается новое значение. В словарь переходов *roots* добавляется пара из клетка - родитель.

fullIteration() – метод используется для полного выполнения алгоритма (без рассмотрения каждой конкретной итерации). Так же из очереди извлекается клетка с наименьшим приоритетом и рассматриваются её соседи. В результате работы метод возвращает словарь переходов *roots.*

recoverPath(MutableMap<Cell, Cell?>) – метод восстанавливает путь от старта до финиша на основе переданного словаря переходов.

Для реализации очереди с приоритетом, представленной в виде минимальной двоичной кучи, был написан класс *Heap*:

Основные методы:

siftUp(index) – метод, осуществляющий просеивание элемента с индексом *index* вверх.

siftDown(index) – метод, осуществляющий просеивание элемента с индексом *index* вниз.

extractMin() – метод, извлекающий минимальный элемент из кучи. Первый и последний элементы меняются местами, после чего последний (бывший первый) удаляется из кучи, а первый (бывший последний) просеивается вниз.

put(element) – метод, помещает элемент в кучу. Изначально элемент добавляется в конец, после чего просеивается вверх.

size() – метод возвращает длину списка, формирующего кучу.

Field – класс, хранящий в себе двоичный массив клеток, координаты старта и финиша.

Основные методы:

init – конструктор, единственная задача которого - внесение координат клеток в их поля.

default – метод, устанавливающий в поля всех клеток начальное значение.

Cell – ячейка карты, хранящая свой тип местности, состояние и числовые характеристики (длину пути до клетки, эвристическая оценка и приоритет).

Основные методы:

setParams(g, h) – метод принимает на вход длину пути до клетки (g) и эвристическую оценку (h), после чего вычисляет приоритет (g + h = f) и помещает все 3 числа в поля клетки.

getWeight() – возвращает стоимость перехода в клетку, основываясь на типе местности клетки.

changeBase() – меняет тип местности клетки на следующий по порядку.

changeEdge(string) – устанавливает в клетку старт/финиш, основываясь на переданной в качестве аргумента строке.

3.2. Для отображения поля

Реализованы классы *CellView* и *FieldView*.

CellView служит для отрисовки ячеек.

Основные методы:

makeBox(cell) – в соответствии с типом ячейки (трава / вода / камень), ее состоянием (не обрабатывалась / обрабатывается / уже обработана / содержится в пути) и статусом (финиш / старт / обычная ячейка) рисует для нее текстуру. Выводит численные значения ячейки g, h, f.

screenInformationAboutTypes() – выводит для справки вид ячеек в правой части приложения.

FieldView нужен для отрисовки всего поля.

Содержит метод *drawField(field, cellView)* который для каждой ячейки поля вызывает метод *cellView.makeBox* для ее отрисовки.

3.3. Для интерфейса приложения

Реализован класс *Controller* для взаимодействия между пользователем и программой.

Основные методы:

userInputCord() – выводит окно в котором можно ввести размеры поля самому или ввести все нужные данные из файла формата .txt.

algorithmScreen() – выводит главное окно, в котором будет запускаться алгоритм. В этом методе вызываются все главные функции в запуске и работе алгоритма.

helpDialog(onDismiss: () -> Unit) – выводит диалоговое окно с помощью, в котором написано описание алгоритма.

errorAlert(*onDismiss: () -> Unit, message: String)* – выводит диалоговое окно с ошибкой, которую мог указать пользователь.

defaultSettings() – метод, который устанавливает всем полям классов, которые участвуют в работе алгоритма значения по-умолчанию.

Класс *FileReader* используется для считывания поля из файла.

Класс *Logger* и *Singleton* для передачи сообщений из алгоритма в окно приложения.

3.4. Итоговая иерархия классов

Uml-диаграмма представлена на рисунке 4:

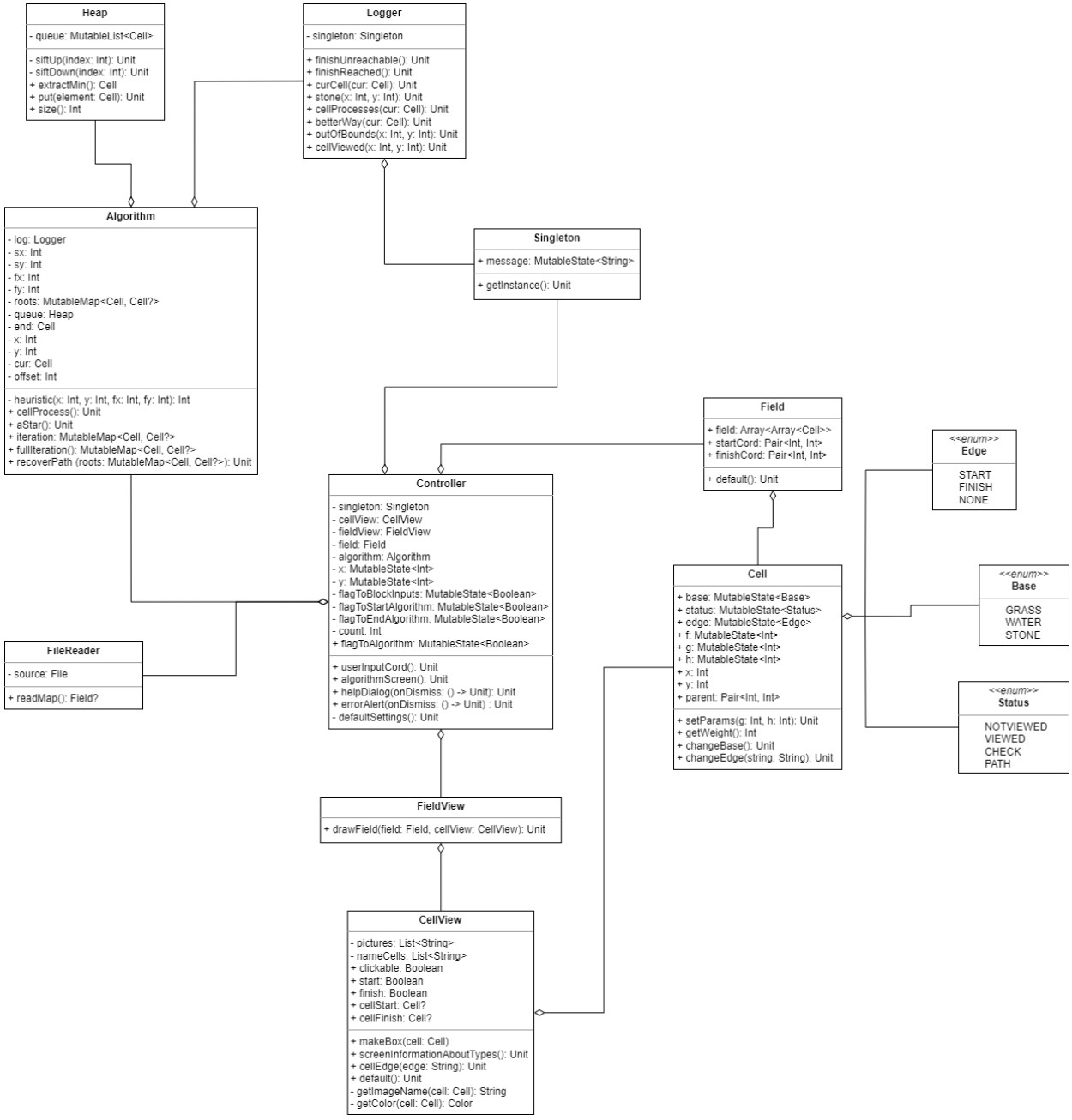

Рисунок 4 – Конечная иерархия классов

4. ТЕСТИРОВАНИЕ

4.1. Тестирование интерфейса и обработки исключительных ситуаций.

Рассмотрим набор исключительных ситуаций и реакцию программы на них:

1. На рисунке 5 представлена реакция программы на некорректное задание размеров поля или стартовой/конечной клеток.

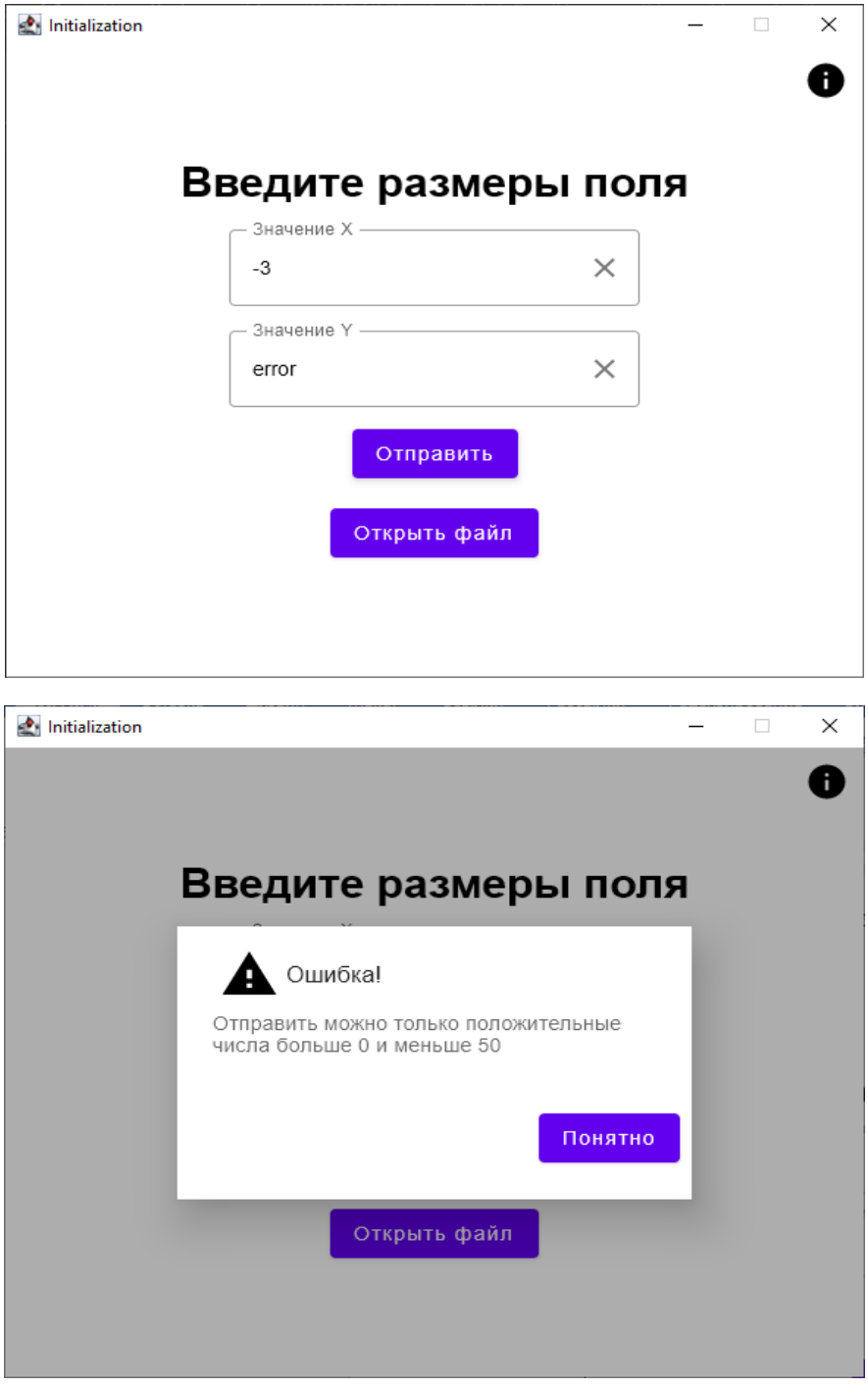

Рисунок 5 – Ошибка при считывании некорректных размеров поля.

Ошибка выбрасывается в случае, если было введено что-то кроме положительного целого числа. При считывании из файла тот же принцип распространяется на размеры поля, но также проверяется, не выходят ли считанные координаты старта и финиша за границы.

2. На рисунке 6 представлена ситуация, когда размеры считаны правильно, но пользователь не установил на поле старт и финиш. В таком случае эти ключевые точки будут установлены в значения по умолчанию: старт – в левый верхний угол, финиш – в правый нижний.

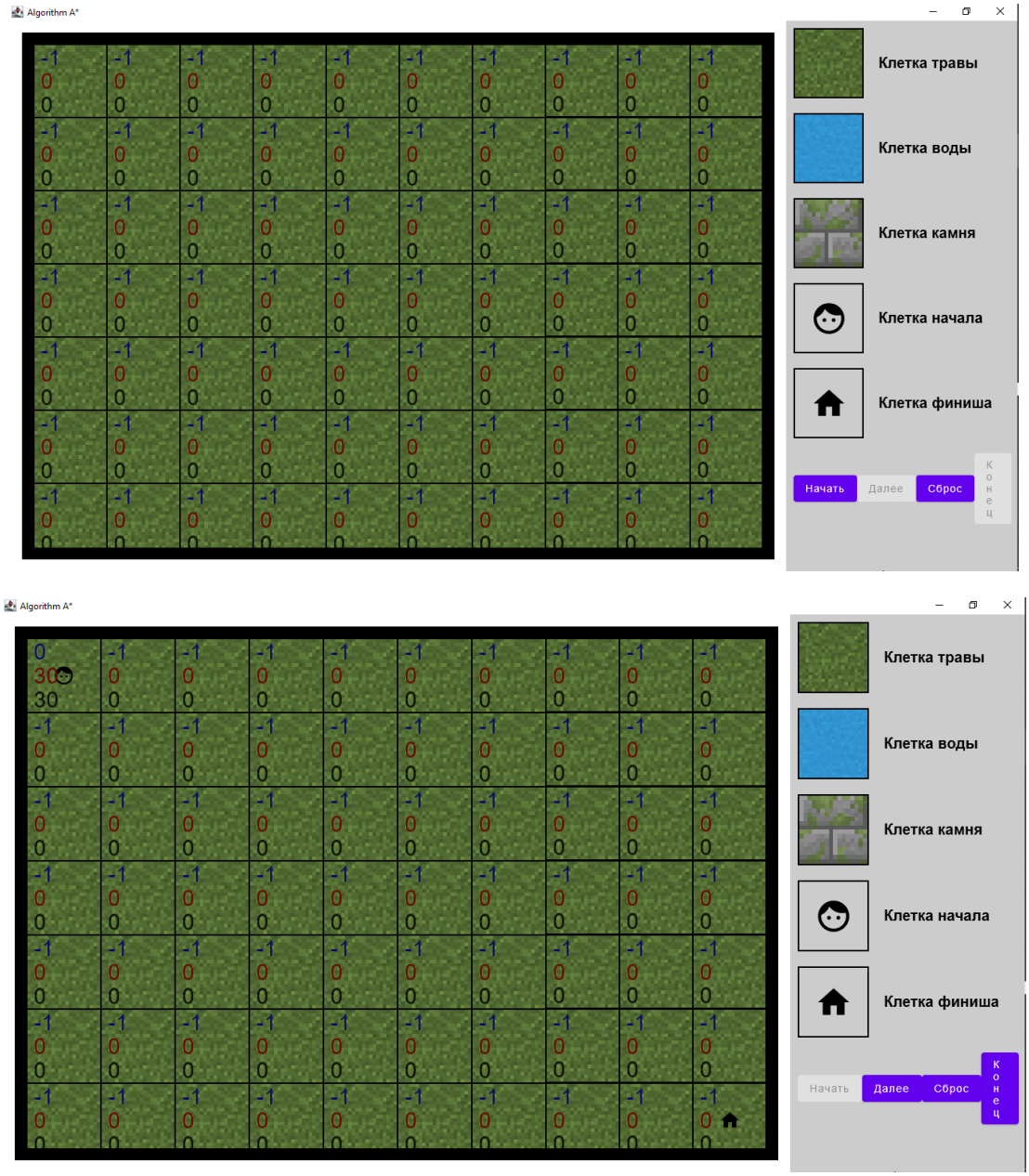

Рисунок 6 – Установка старта и финиша по умолчанию.

Несложно видеть, что программа автоматически установила недостающие старт и финиш. В том случае, когда была установлена одна из крайних клеток, вторая также будет установлена в позицию по умолчанию.

3. Работа программы в ситуации, когда из файла корректно считаны только размеры поля, а также координаты старта и финиша, представлена на рисунке 7.

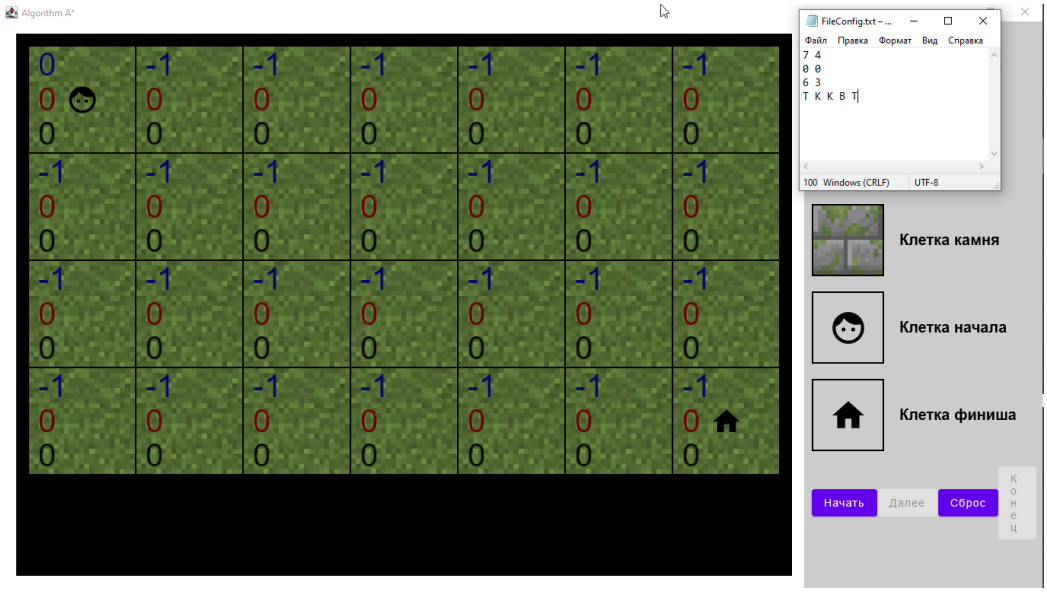

Рисунок 7 - Неполная конфигурация клеток поля в файле.

Незаданные клетки автоматически будут покрыты травой.

4. На рисунке 8 представлена ситуация, когда в файле отсутствуют данные о поле/старте/финише.

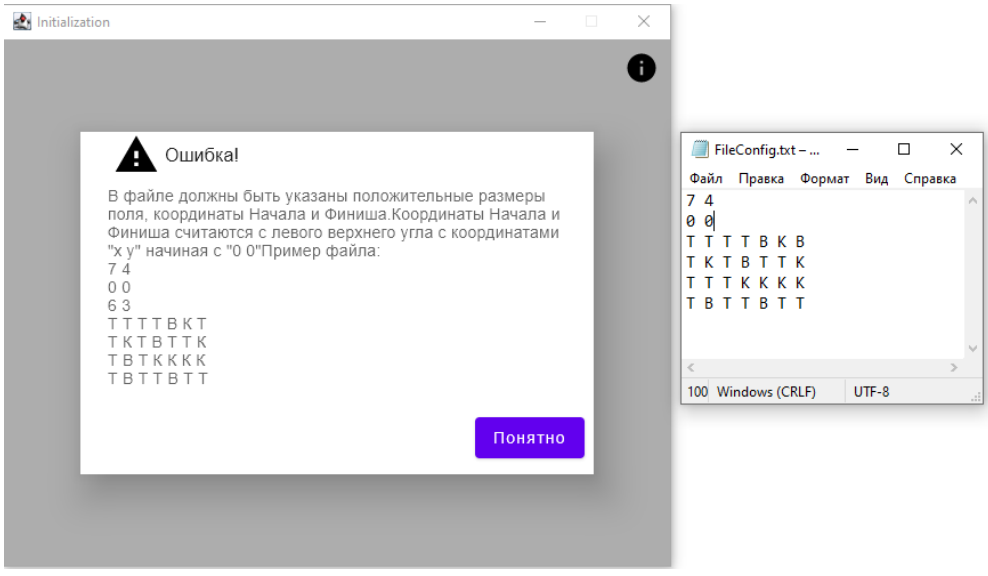

Рисунок 8 – Отсутствие данных о финише в файле.

В файле не хватает данных о координатах финиша. Программа обрабатывает это и предлагает пользователю пример корректной конфигурации.

5. Рисунок 9 иллюстрирует работу программы, когда пользователь пытается установить два старта.

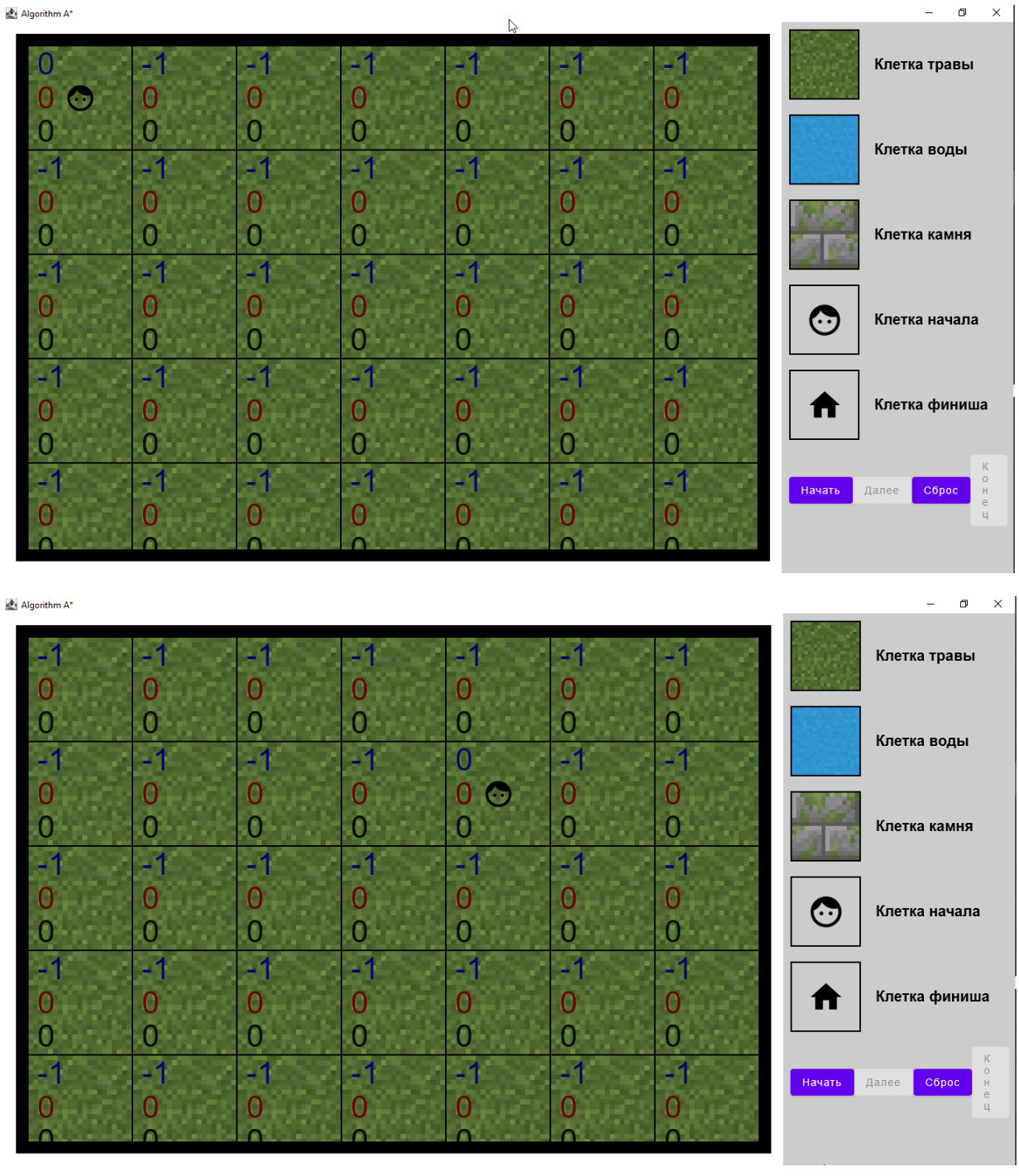

Рисунок 9 – Установка двух стартов.

В случае, если на поле уже есть старт, попытка установить ещё один приведёт к удалению уже имеющегося и установке нового. Аналогичная ситуация произойдет при установке финиша.

6. На рисунке 10 показана реакция программы на попытку пользователя установить старт на финиш.

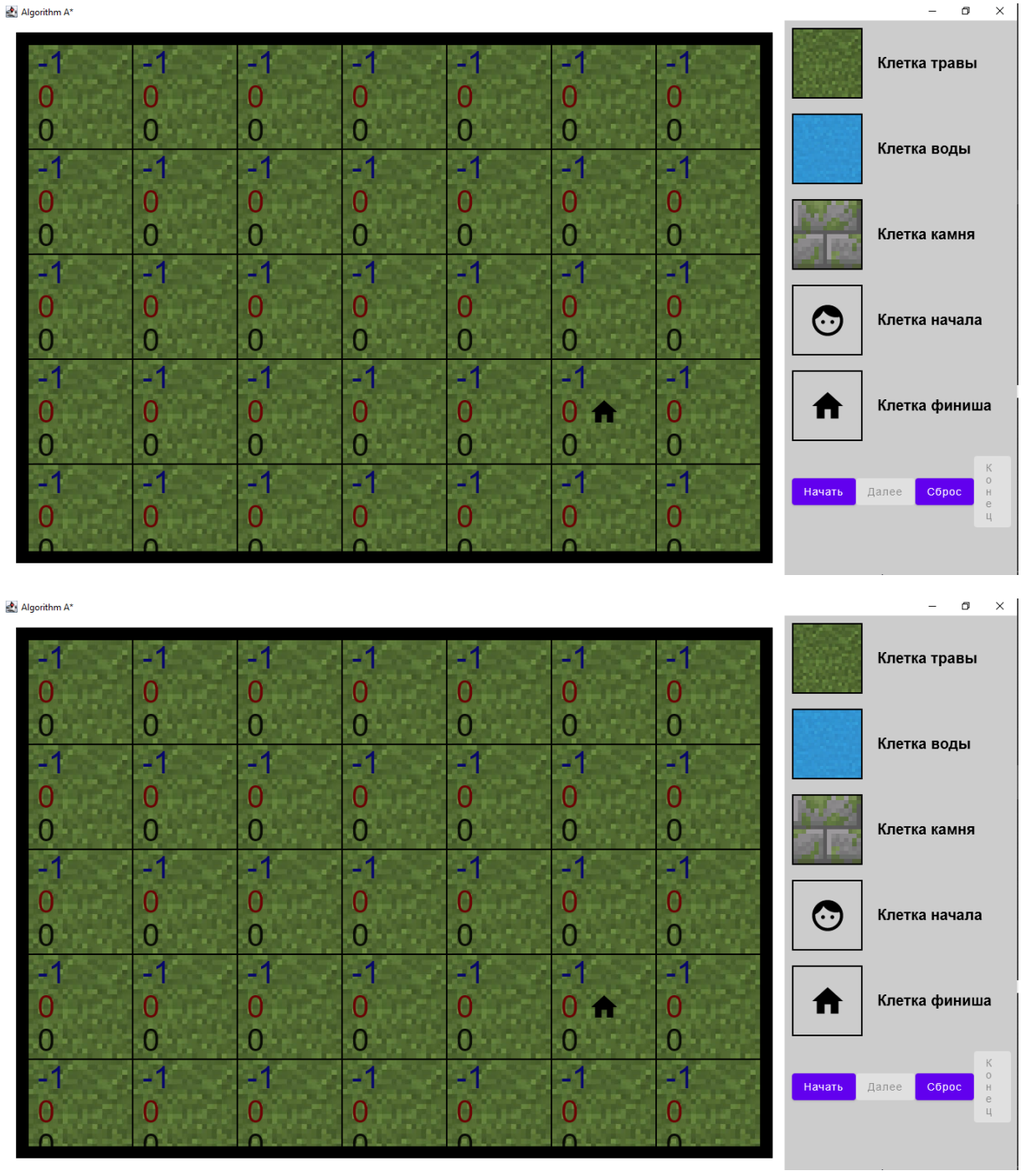

Рисунок 10 – Установка старта на финиш.

Видно, что пользователь без проблем может установить старт на финиш, при этом последний будет удалён, и при запуске программы будет установлен в

позицию по умолчанию. Однако, при попытке установить финиш на старт, у нас ничего не выйдет, потому как у старта есть приоритет по установке (старт можно поставить в любом случае, финиш можно поставить, если на клетке не установлен старт).

4.2. Тестирование алгоритма.

1. Рисунок 11 представляет результат работы программы на обычной карте с препятствиями.

Рисунок 11 – Запуск программы на карте с препятствиями.

2. Рисунок 12 иллюстрирует результат работы программы на карте с недостижимым финишем.

Рисунок 12 – Запуск программы на карте с недостижимым финишем.

3. На рисунке 13 представлена работы программы на карте без препятствий.

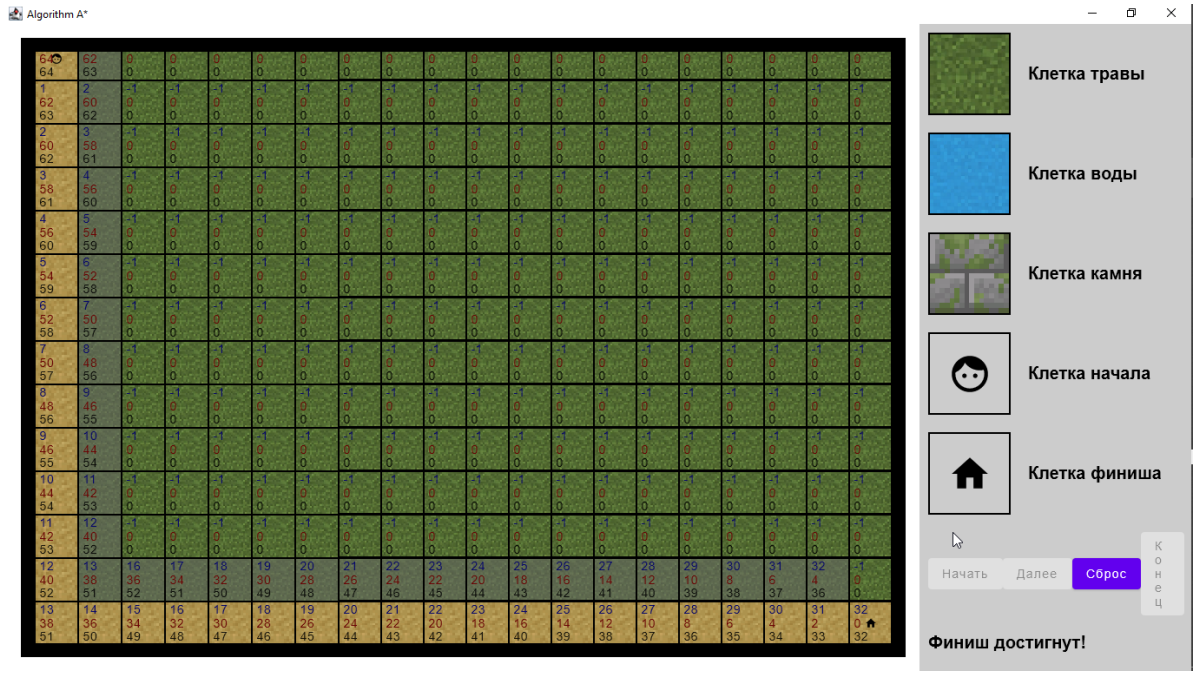

Рисунок 13 – Запуск программы на карте без препятствий.

4. Рисунок 14 показывает результат работы программы на карте-лабиринте.

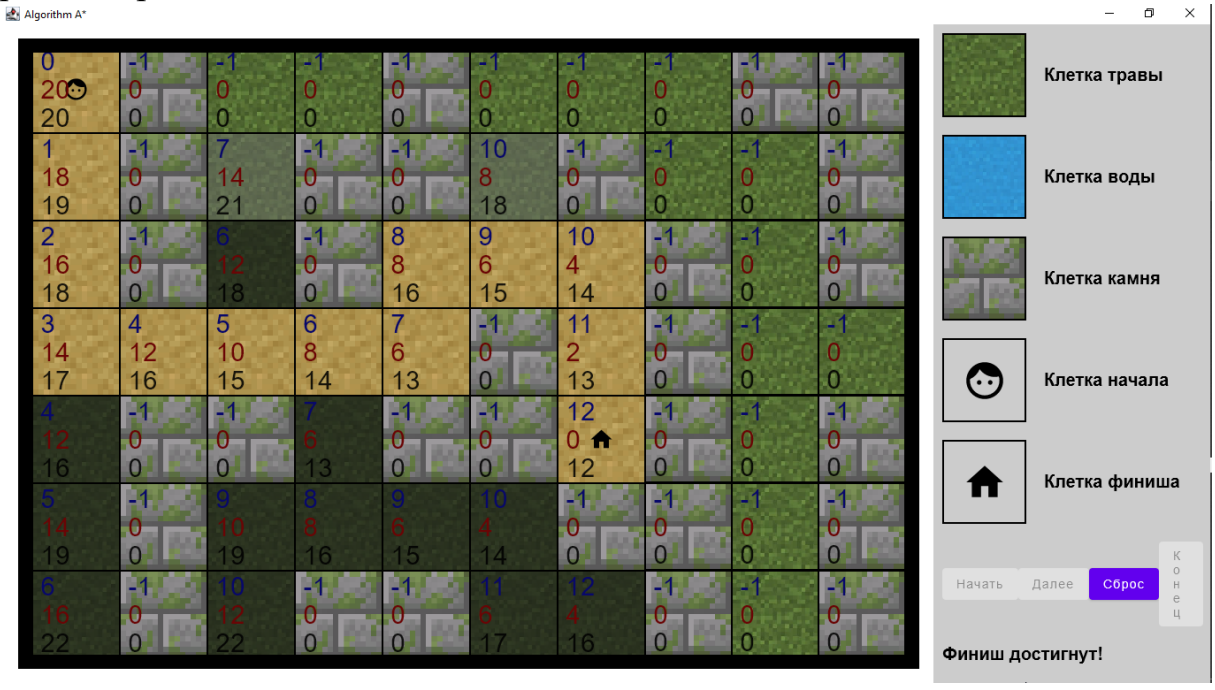

Рисунок 14 – Запуск программы на лабиринте.

ЗАКЛЮЧЕНИЕ

В ходе выполнения практической работы было реализовано приложение с графическим интерфейсом, демонстрирующее пошаговое выполнение алгоритма A*. Закреплены навыки программирования на языке Kotlin.

Для написания GUI была изучена библиотека Compose Jetpack.

Итоговая программа соответствует требованиям, предъявленным в начале работы.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сайт Alexanderklimov.

URL: https://developer.alexanderklimov.ru/android/simplepaint.php

2. Сайт kotlinlang.

URL: https://kotlinlang.ru/docs/reflection.html

3. Сайт metanit.

URL: https://metanit.com/kotlin/jetpack

4. Сайт android.

URL: https://developer.android.com/jetpack/compose

5. Репозиторий бригады.

URL: https://github.com/KirillSamokhin/Brigade_4.git