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

ОТЧЁТ

по учебной практике

Тема: Копатель оффлайн

Студент гр. 8381 _____ Почаев Н.А.

Студентка гр. 8381 _____ Лисок М.А.

Студент гр. 8381 _____ Перелыгин Д.С.

Руководитель _____ Фирсов М.А.

Санкт-Петербург
2020

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Почаев Н.А. группы 8381
Студентка Лисок М.А. группы 8381
Студент Перелыгин Д.С. группы 8381

Тема практики: применение алгоритмы Дейкстры для поиска кратчайшего пути на 2D карте в контексте задачи

Задание на практику:

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

Алгоритм: алгоритм Дейкстры.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 11.07.2020

Дата защиты отчета: 11.07.2020

Студент	_____	Почаев Н.А.
Студентка	_____	Лисок М.А.
Студент	_____	Перелыгин Д.С.
Руководитель	_____	Фирсов М.А.

АННОТАЦИЯ

Целью выполнения данной работы является практическое применения алгоритмы Дейкстры. Созданная программа выполняет визуализацию двумерной карты, состоящей из блоков разных типов - руды, выделенные типы которых образуют зоны с 0-ой стоимостью прохода — жила, когда проход по основной руде образует стоимость рёбер между жилами. Задачей является нахождение оптимального пути персонажа с поверхности (точка $(0,0)$) до случайно сгенерированной точки на нижнем уровне — портала.

SUMMARY

The purpose of this work is a practical application of the Dijkstra algorithm. Created the program performs visualization of two-dimensional maps consisting of different ore types selected which form a zone with a 0-th cost of passage — lived when a pass is on the main ore forms the cost of edges between cores. The task is to find the optimal path of the character from the surface (point) to a randomly generated point on the lower level — the portal.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На вход программе подаётся карта руд — блоков разных видов, а также земли и целевой точки — портала. Передвижение между соединёнными рудами в 4-х направлениях является «бесплатным», назовём такие участки жилами. Передвижение же между ними по блокам земли обладаем стоимостью: 1 у.е. за один блок. Целью является за минимальную стоимость довести персонажа из начальной точки $(0,0)$ в финальную.

Для решения задачи применяется алгоритм Дейкстры. В начале работы программы находятся все жилы, представленные на карте — они являются вершинами обрабатываемого графа. Затем ищутся тривиальные пути между всеми вершинами. На полученном полном графе запускается алгоритм Дейкстры, который находит наикротчайший путь между начальным местоположением персонажа и порталом, используя факт «бесплатного» прохождения внутри жил (путь улучшаются за счёт нахождения промежуточных жил).

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требования к входным данным

Рабочая карта может быть получена 2-мя путями: через загрузку из файла JSON формата, указанного ниже, либо через генерацию случайно карты, путём нажания Edit-Generate.

Формат входного файла имеет вид JSON массива, где каждый элемент — клатка на поле, представленная в следующем виде:

```
{
        "posX" : 0,
        "posY" : 0,
        "checked" : false,
        "ore" :
            {
                "imgFileName" : "ground.jpg",
                "oreType" : "GROUND"
            }
},
```

Поля imgFileName и checked являются опциональными — они будут распознаны при десерелизации файла карты, однако, как будет отдельного отмечено далее, не участвуют в серелизации.

1.1.2. Требования к визуализации

Карта должна выводиться на экран и визуализировать различные виды руд, а также текущее местонахождение персонажа, если используется пошаговый режим или же итоговый путь в конце / при использовании опции Just Result.

1.1.3. Требования к выходным данным

Выходные данный аналогично представляются файлов формата JSON, данный в котором имеют следующий вид:

```
{
        "posX" : 0,
        "posY" : 0,
        "ore" :
            {
                "oreType" : "GROUND"
            }
},
```

Поля checked и imgFileName являются либо локально используемыми для алгоритма, либо неконстантыми, в случае названия файла для визуализации. Таким образом, в серелизации они участвовать не должны.

1.2. Уточнение требований после консультации с преподавателем

По итогу консультации с преподавателем были отмечены следующие замечания, учтённые в последующих версиях программы:

- Неравномерное отображение линий разметки поля по горизонтали при выводе изображений с рудой. Было исправлено путём изменения форматирования вёрстки макета программы.
- В начальной версии формата входых данных содержалось название файлов с картинками, что являлось негибким решением, непозволяющем в будущем гибко менять текстуры приложения. Было исправлено, путём уточнения полей серелизации, как было описано ранее.
- Была выдвинута необходимость тестирования входных данных некоректность. Данное требование было удовлетворено путём использования механизма исключений и провода транзакции при считывании файла: в случае наличия некорректых значений, данные программы приводятся в начальное состояние, а поле визуализации очищается.

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

К 02.07.20 должны быть распределены роли между членами бригады, определён функционал программы, создана начальная версия UML и прототип интерфейса.

К 04.07.20 должна быть создана разметка интерфейса с основным функциональными элементами, выполнены соединения с прототипами бизнес логики для реагирования на события.

К 06.07.20 должна быть разработана система загрузки и сохранения рабочих карт, их генерация, а также визуализация в интерфейсе программы. Также к данному этапу должны быть разработаны алгоритмы выделения жил на карте, а также нахождения весам рёбер между ними.

К 10.07.20 должен быть завершён алгоритм обработки карты (бекенд часть) с API для получения массива данных, подающихся визуализации. Также должны быть разработаны механизмы пошагового вывода результата и очистки рабочей области программы.

К 11.07.20 должно быть завершено соединение бизнес логики и GUI программы, проведено тестировани функциональных классов программы.

2.2. Распределение ролей в бригаде

- Почаев Н.А. - лидер, фронтенд, документация, бэкенд функционирования программы, тестирование.
- Лисок М.А. - алгоритмист, документация, тестирование.
- Перелыгин Д.С. - тестирование, реализация некоторых аспектов бэкенд, документация.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Структуры данных

Таблица 1 – Основные структуры данных, необходимые для поиска кратчайшего пути

Ore	Класс, который описывает тип руды и путь к картинке, изображающей данный тип
OreTypes	Перечисление, описывающие все типы руд, которые могут быть на поле
Cell	Класс, описывающий позицию клетки и тип руды, который клетка может принимать
Field	Класс поля состоит из двумерного массива клеток. Имеет определенную высоту и ширину
Node	Класс, описывающий элемент графа. Содержит номер руды, к которой принадлежит, клетку из которой пришли в данный элемент, клетку самого элемента, предыдущий элемент, для восстановления кратчайшего пути, вес ребра к данному элементу, минимальный путь к данному элементу от точки старта
Graph	Основной класс, описывающий граф. Содержит поле, массив руд с номером руды и клетками принадлежащими каждой, клетку и номер старта, клетку и номер ада, минимальный путь от старта до ада, клетки и элементы, входящие в кратчайший путь

3.2. Основные методы

Таблица 2 – Основные методы класса Graph

Название	**Назначение**
findVein(int x, int y, OreTypes type, ArrayList<Cell> vein)	Параметры: • int x, y – позиция клетки, с которой начинается новая жила • OreTypes type – тип руды новой жилы • ArrayList<Cell> vein – массив, в который будут записаны все клетки принадлежащие одной жиле Метод ищет все клетки принадлежащие одной жиле
findVeins()	Находит все жилы на поле

findMinWayBetweenTwoVeins(ArrayList<Cell> vein1, ArrayList<Cell> vein2, Integer numberVein1, Integer numberVein2)	Параметры: • ArrayList<Cell> vein1 – массив клеток жилы, из которой ищем кратчайший путь • ArrayList<Cell> vein2 – массив клеток жилы, в которую ищем кратчайший путь • Integer numberVein1 – номер жилы, из которой ищем кратчайший путь • Integer numberVein2 – номер жилы, в которую ищем кратчайший путь Метод ищет кратчайший путь между двумя жилами, сравнивая поочередно расстояния между всеми клетками жил
findMinWayBetweenVeins()	Находит кратчайшие пути между всеми жилами
setNodesToGraph()	Добавляет элементы в граф
setCellsOfGround(Cell from, Cell to)	Параметры: • Cell from – клетка выхода из жилы • Cell to - клетка входа в жилу Добавляет в массив клеток кратчайшего пути клетки земли, которые встречаются при переходе из одной жилы в другую
dijkstra()	Реализует работу алгоритма Дейкстры
shortestWay()	Восстанавливает кратчайший путь

	полученный в результате работы алгоритма Дейкстры
getCellsOfShortestWay()	Возвращает все клетки, которые входят в кратчайший путь. Кратчайший путь – путь от точки старта до точки ада В состав клеток входят: • клетки входа/выхода в/из жил • клетки земли, для перехода из одной жилы в следующую • клетка старта • клетка ада
getVeins()	Возвращает все жилы расположенные на поле
getGraph()	Возвращает переменную HashMap<Integer, ArrayList<Node>> graph, которая хранит в себе номер жилы и все клетки принадлежащие данному номеру жилы
getLengthMinWay()	Возвращает длину минимального пути от точки старта до точки ада

3.3. Особенности бэкенд, технологии и библиотеки

Как уже было отмечено, сохранения и загрузка карт выполняется при помощи использования формата хранения текстовых данных — JSON. Для серилизации и десерилизации объектов клеток карты используется библиотека Google-GSON. В ней применяются специфичные методы, такие как, например, setPrettyPrinting() используется для читабельного представления выходных данных JSON и excludeFieldsWithoutExposeAnnotation() для выбора полей, используемых для серилизации и десерелизации.

Для визуализации используется платформа Java FX, а также встроенные технологии в виде формата разметки fxml и специализированных тегов CSS для

стилизиции определённых элементов интерфейса. Для проектирования интерфейса также использовалась программа Scene Builder.

Также в ходе работы, проводимой в программе IntelliJ IDEA было выполнено подключение специализированной библиотеки с аннотациями org.jetbrains.org.jetbrains для выполнения дополнительных проверок в ходе работы. Таких как, например, @NotNull, указывающих, что значения не должно принимать значение null.

Для контроля зависимых библиотек, удобного доступа к ресурсам и автоматизированной сборки проекта используется фреймворк Maven. Он преднозначен для декларативного описания проекта. Также использования было обусловлено необходимостью ведения кроссплатформенной разработки - участники бригады работают на системах: Mac OS, Linux Manjaro и Windows 7.

3.4. Особенности визуализации

Как было отмечено ранее, для визуализации используетяся платформа Java FX. В данном проекте разметка основного окна находится в файле miner.fxml, а связь с бизнес логикой происходит посредством идентификатора id и класса Controller.

Основым объектов сцены визуализации является GridPane и фиксированными размерами клетки и выводом их границ. Через реальизованный класс ImageCell происходит маштабирование изображения руд для визуализации.

4. ТЕСТИРОВАНИЕ

4.1. Unit тестирование программы

Для автоматической проверки корректности работы алгоритмов была использована JUnit4. Unit тесты покрывают основные методы классов Graph и MapGenerator.

В классе Graph в первую очередь проверяется правильность работы алгоритма и вспомогательных методов, позволяющих выполнить корректный поиск пути. В частности, кроме конечного ответа проверяется количество жил руды, найденных методом findVeins, а так же местоположение клетки портала, кратчайший путь до которой необходимо найти.

Для класс MapGenerator тестируются методы generateMap и dirtAndAirFiller. Unit тестирование осуществляет проверку самых необходимых параметров для успешного выполнения алгоритма и визуализации для случайно сгенерированных карт, а именно: постановку портала, отсутствие пустых клеток при генерации и наличие на карте рудных жил.

4.2. Мануальное тестирование программы

Ручное тестирование программы выполнялось на сгенерированных картах, чтобы была возможность проверить корректность программы на случайных тестах, большее внимание было уделено нетривиальным случаям, чтобы убедиться в надежности алгоритма. Так же были протестирован функционал программы, включающий в себя загрузку и сохранения файлов карт, пошаговое выполнение алгоритма, корректность визуализации.

На рисунках 1 и 2 продемонстрированы случаи, когда путь на первый взгляд не очевиден, однако алгоритм верно находит кратчайший.

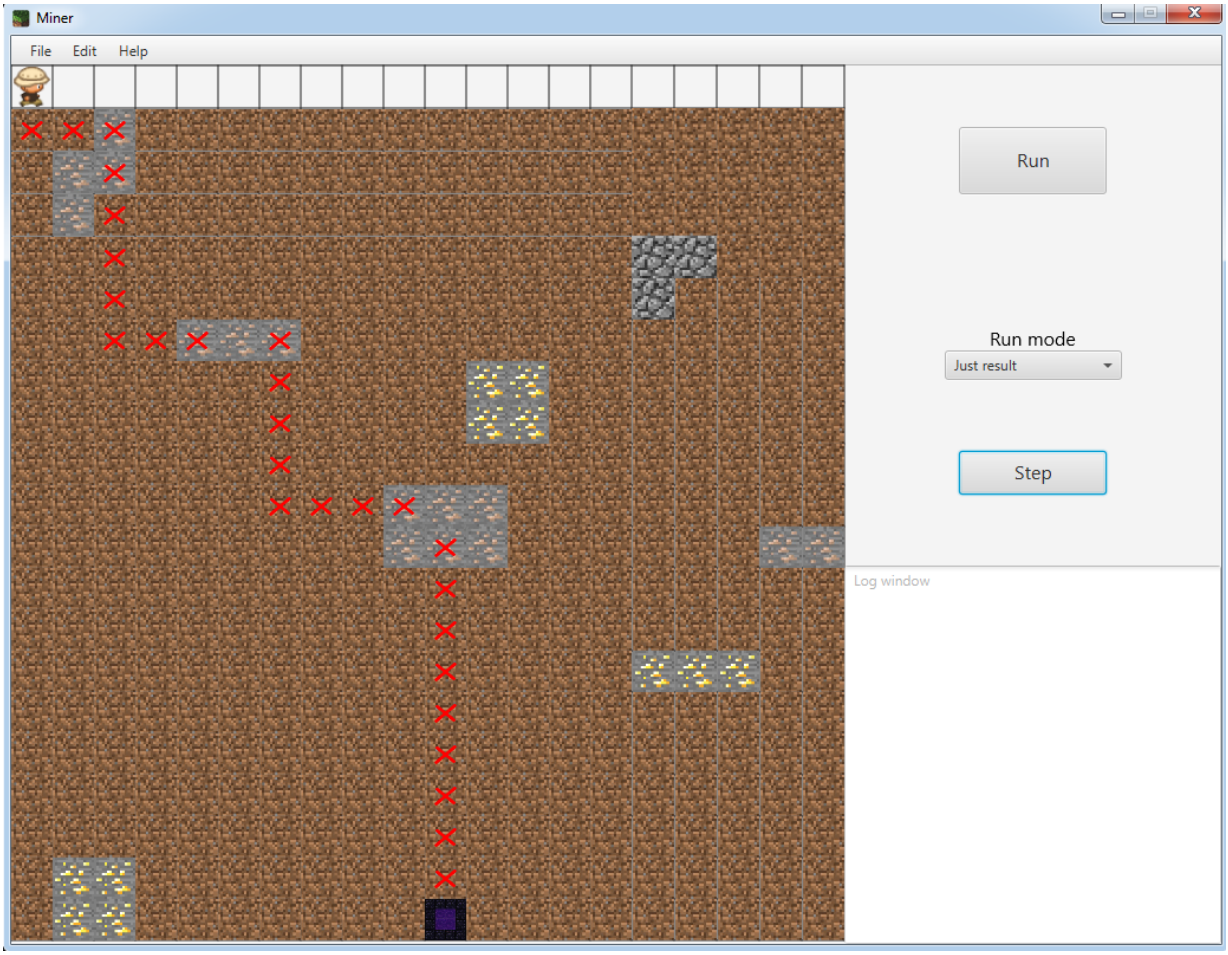


Рисунок 1. Не очевидный кратчайший путь.

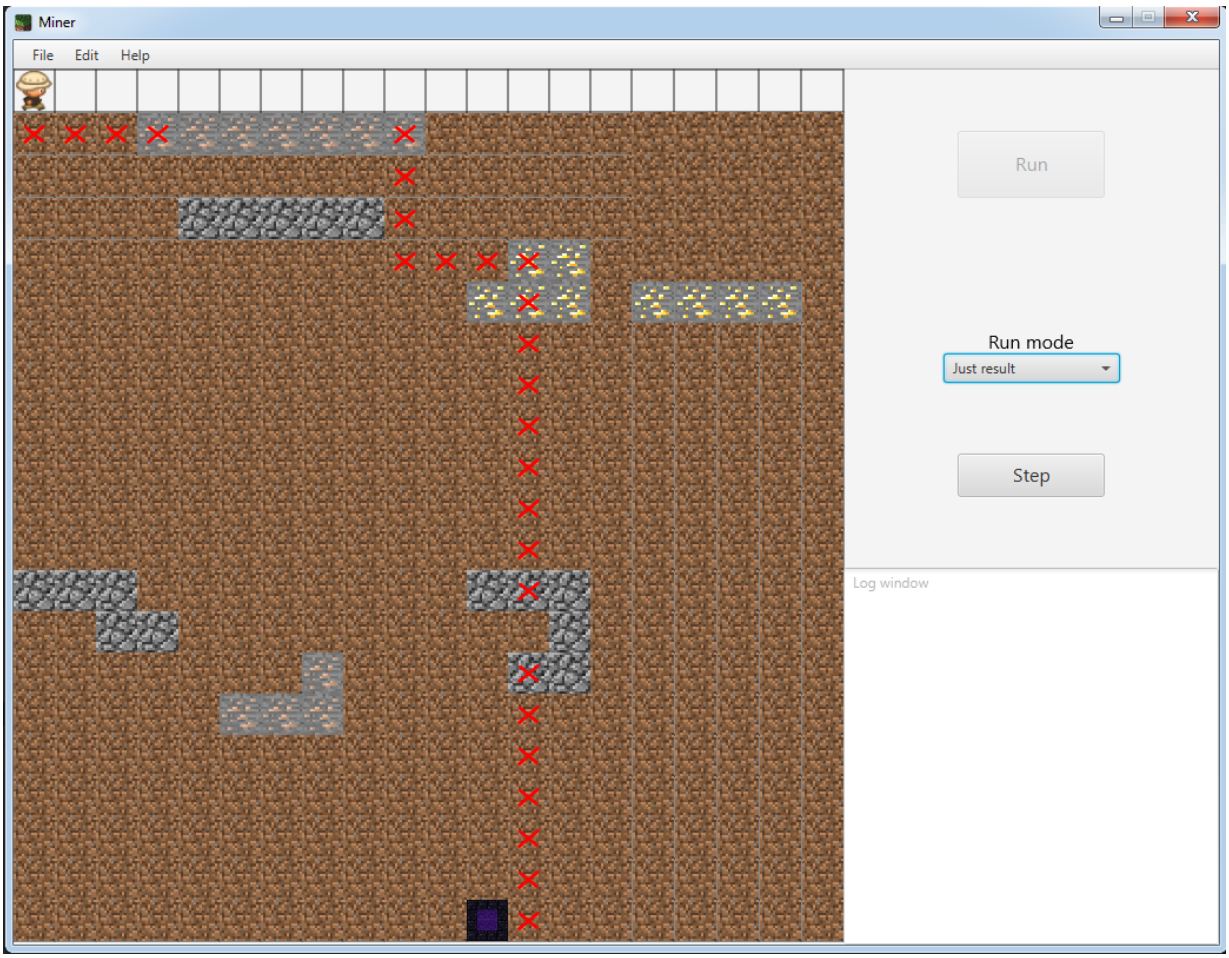


Рисунок 2. Нетривиальный оптимальный путь.

ЗАКЛЮЧЕНИЕ

Проект был завершен успешно. Были выполнены поставленные цели, содержащиеся в техническом задании. Был реализован удобный пользовательский интерфейс, упрощающий взаимодействие с программой, а также наглядная визуализация процесса поиска кратчайшего пути, в том числе и пошаговая. Функционал приложения включает в себя: загрузку и сохранения карты, возможность сгенерировать случайную карту и очистить окно визуализации в случае необходимости.

Работа программы основывается на алгоритме Дейкстры, осуществляющий поиск кратчайшего пути между вершинами в графе.

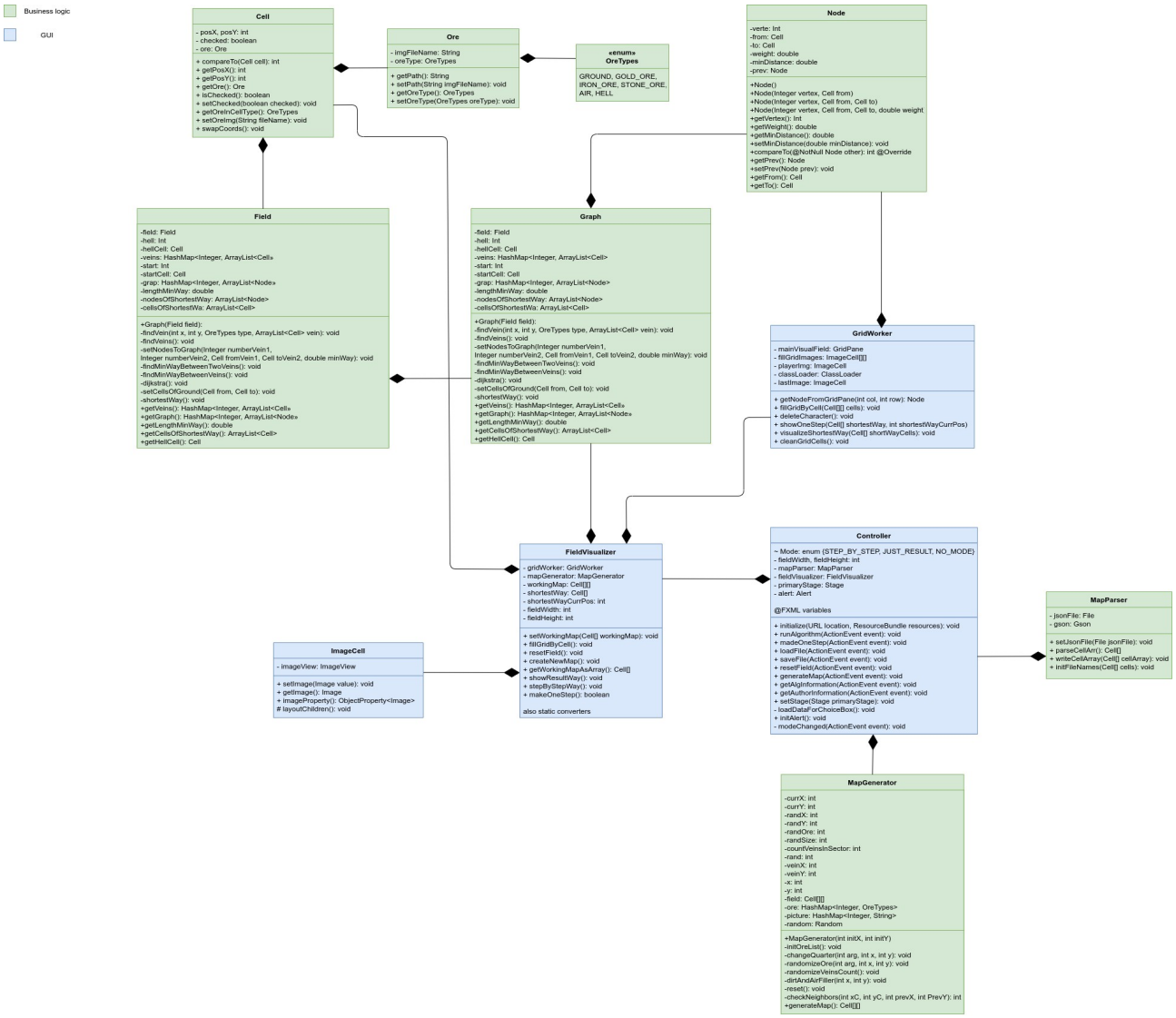
Работоспособность алгоритма была проверена тестированием, использующим случайно сгенерированные карты, чтобы рассмотреть все возможные случаи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Шилдт Герберт. Java. Полное руководство.: Диалектика, 2018
2. Pro JavaFX 8 - A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients / Weaver, J., Gao, W., Chin, S., Iverson, D., Vos, J.: Apress, 2018
3. Бхаргава Адитья. Грокаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих.: Питер, 2019
4. Кент Бек. JUnit Pocket Guide.: O`RELLY, 2014
5. Онлайн учебник по Java FX, URL: https://o7planning.org/ (дата обращения: 06.08.2020)

ПРИЛОЖЕНИЕ Б

UML ДИАГРАММА



ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл Main.java

```
package etu.leti.gui;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.image.Image;
import javafx.stage.Stage;
import org.jetbrains.annotations.NotNull;

import java.net.URL;

public class Main extends Application {

    Stage window;

    /**
     *
     * @param primaryStage
     * The surrounding window, which is used as the initial canvas
and contains the remaining components
     * @throws Exception
     * For load method
     */
    @Override
    public void start(@NotNull Stage primaryStage) throws
Exception{
        window = primaryStage;

        URL scene =
getClass().getClassLoader().getResource("canvas/miner.fxml");
        assert scene != null;

        FXMLLoader fxmlLoader = new FXMLLoader(scene);
        Parent root = (Parent) fxmlLoader.load();
        ((Controller)
fxmlLoader.getController()).setStage(primaryStage);

        primaryStage.setTitle("Miner");
        primaryStage.setScene(new Scene(root));

        // Block resizing
        primaryStage.setMaxHeight(primaryStage.getHeight());
        primaryStage.setMaxWidth(primaryStage.getWidth());
        primaryStage.setResizable(false);
```

```
        // Set application icon
        primaryStage.getIcons().add(new Image("img/icon.png"));

        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Файл FieldVisualizer.java

```
package etu.leti.gui;

import etu.leti.algorithm.Graph;
import etu.leti.field.Field;
import etu.leti.generator.MapGenerator;
import javafx.scene.layout.GridPane;
import org.jetbrains.annotations.NotNull;

import etu.leti.field.Cell;
import etu.leti.gui.gridhandler.GridWorker;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.List;

public class FieldVisualizer {

    private final GridWorker gridWorker;
    private final MapGenerator mapGenerator;

    private Cell[][] workingMap;
    private Cell[] shortestWay;
    private int shortestWayCurrPos;
    private int fieldWidth;
    private int fieldHeight;

    public FieldVisualizer(GridPane mainVisualField, ClassLoader
classLoader, int fieldWidth, int fieldHeight) {
        gridWorker = new GridWorker(mainVisualField, classLoader);
        mapGenerator = new MapGenerator(fieldWidth, fieldHeight);
        this.fieldWidth = fieldWidth;
        this.fieldHeight = fieldHeight;
        workingMap = null;
        shortestWay = null;
        shortestWayCurrPos = 0;
    }

    public Cell[][] getWorkingMap() {
```

```
        return workingMap;
    }

    public void setWorkingMap(Cell[][] workingMap) {
        this.workingMap = workingMap;
    }

    public void setWorkingMap(Cell[] workingMap) {
        this.workingMap = convert1DArrayTo2D(workingMap,
fieldWidth, fieldHeight);
    }

    public void fillGridByCell() {
        gridWorker.fillGridByCell(workingMap);
    }

    public void resetField() {
        gridWorker.cleanGridCells();
    }

    public void setSize(int width, int height) {
        fieldWidth = width;
        fieldHeight = height;
    }

    public void createNewMap() {
        workingMap = mapGenerator.generateMap();
        resetField();
        mapGenerator.reset();
        fillGridByCell();
    }

    public Cell[] getWorkingMapAsArray() {
        return convert2DArrayTo1D(workingMap);
    }

    public void showResultWay() {
        Cell[][] convertedArray =
convertArrayForAlgorithm(convertCoordsOf2DArray(workingMap));
        Graph graph = new Graph(new Field(convertedArray,
fieldHeight, fieldWidth));
        ArrayList<Cell> result = graph.getCellsOfShortestWay();
        shortestWay = result.toArray(new Cell[0]);

gridWorker.visualizeShortestWay(convertCoordsOf1DArray(shortestWay
));
        shortestWay = null;
    }

    public void stepByStepWay() {
        Cell[][] convertedArray =
convertArrayForAlgorithm(convertCoordsOf2DArray(workingMap));
```

```
        Graph graph = new Graph(new Field(convertedArray,
fieldHeight, fieldWidth));
        ArrayList<Cell> result = graph.getCellsOfShortestWay();
        shortestWay = result.toArray(new Cell[0]);
        shortestWay = convertCoordsOf1DArray(shortestWay);
        // begin - top and left; end - bottom and right
        Arrays.sort(shortestWay);
        shortestWayCurrPos = shortestWay.length - 1;
    }

    public boolean makeOneStep() {
        if(shortestWayCurrPos == -1) {
            return true;
        }

        gridWorker.showOneStep(shortestWay, shortestWayCurrPos);
        shortestWayCurrPos--;

        return false;
    }

    /**
     * @param cells
     * TWO dimensions array of Cell
     * @return
     * NEW TWO dimensions array of Cell with REVERSED X and Y
coords
     */
    public static Cell[] @NotNull [] convertCoordsOf2DArray(Cell[]
@NotNull [] cells) {
        Cell[][] resultCells = new Cell[cells.length][];
        for(int i = 0; i < cells.length; ++i) {
            resultCells[i] = Arrays.stream(cells[i]).
                    map(cell -> cell == null ? null : new
Cell(cell)).toArray(Cell[]::new);
        }

        for(Cell[] cellArr : resultCells) {
            for(Cell cell : cellArr) {
                cell.swapCoords();
            }
        }

        return resultCells;
    }

    /**
     * @param cells
     * TWO dimensions array of Cell
     * @return
     * NEW TWO dimensions array of Cell with REVERSED X and Y
coords AND rotated map
```

```
     */
    public static Cell[] @NotNull []
convertArrayForAlgorithm(Cell[] @NotNull [] cells) {
        Cell[][] resultCells = new Cell[cells[0].length]
[cells.length];
        for(Cell[] cellArr : cells) {
            for(Cell cell : cellArr) {
                resultCells[cell.getPosX()][cell.getPosY()] =
cell;
            }
        }

        return resultCells;
    }

    /**
     * @param cells
     * ONE dimensions array of Cell
     * @return
     * NEW ONE dimensions array of Cell with REVERSED X and Y
coords
     */
    public static Cell[] convertCoordsOf1DArray(Cell @NotNull []
cells) {
        Cell[] resultCells = Arrays.stream(cells).
                             map(cell -> cell == null ? null : new
Cell(cell)).toArray(Cell[]::new);
        for(Cell cell : resultCells) {
            cell.swapCoords();
        }

        return resultCells;
    }

    /**
     * @param arr
     * TWO dimensions array of Cell
     * @return
     * ONE dimension array of Cell
     */
    public static Cell @NotNull [] convert2DArrayTo1D(Cell[]
@NotNull [] arr) {
        List<Cell> list = new ArrayList<>();
        for (Cell[] cells : arr) {
            Collections.addAll(list, cells);
        }

        Cell[] finalArray = new Cell[list.size()];
        for (int i = 0; i < finalArray.length; ++i) {
            finalArray[i] = list.get(i);
        }
```

```
        return finalArray;
    }

    /**
     * @param arr
     * ONE dimension array of Cell
     * @return
     * TWO dimensions array of Cell
     */
    public static Cell @NotNull [] @NotNull []
convert1DArrayTo2D(Cell @NotNull [] arr, int width, int height) {
        Cell[][] result = new Cell[width][height];
        int checkCount = 0;
        for(Cell cell : arr) {
            result[cell.getPosX()][cell.getPosY()] = cell;
            checkCount++;
        }
        if(checkCount != height * width) {
            throw new ArrayIndexOutOfBoundsException("There are
not enough elements to convert a " +
                                                     "one-
dimensional array to a two-dimensional one");
        }

        return result;
    }
}
```

Файл ExtraWindowShower.java

```
package etu.leti.gui;

import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.layout.StackPane;
import javafx.scene.text.Text;
import javafx.stage.Modality;
import javafx.stage.Stage;
import org.jetbrains.annotations.NotNull;

import java.io.IOException;

public class ExtraWindowShower {

    public static void showAboutWindow(Stage primaryStage) {
        StackPane authorInfLayout = new StackPane();
        Scene authInfScene = new Scene(authorInfLayout, 230, 100);
        authInfScene.getStylesheets().add("css/author.css");

        Text text = new Text();
```

```
        text.setText("Pochaev Nikita\nMaria Lisok\nDmitry
Perelygin");
        authorInfLayout.getChildren().add(text);

        Stage authInfWindow = new Stage();
        authInfWindow.setTitle("About authors");
        authInfWindow.setScene(authInfScene);

        authInfWindow.initModality(Modality.WINDOW_MODAL);

        authInfWindow.initOwner(primaryStage);

        authInfWindow.setX(primaryStage.getX() + 350);
        authInfWindow.setY(primaryStage.getY() + 350);

        authInfWindow.setMaxHeight(200);
        authInfWindow.setMaxWidth(300);
        authInfWindow.setResizable(false);

        authInfWindow.show();
    }

    public static void showProgInfoWindow(Stage primaryStage,
@NotNull ClassLoader loader) throws IOException {
        Stage algStage = new Stage();
        FXMLLoader fxmlLoader = new
FXMLLoader(loader.getResource("canvas/algorithm.fxml"));

        Parent root = fxmlLoader.load();
        algStage.initModality(Modality.WINDOW_MODAL);
        algStage.setTitle("About algorithm");

        algStage.setScene(new Scene(root));
        algStage.show();
    }
}
```

Файл Controller.java

```
package etu.leti.gui;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;

import com.google.gson.JsonParseException;
import etu.leti.field.Cell;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
```

```
import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.VBox;
import javafx.stage.FileChooser;

import javafx.stage.Modality;
import javafx.stage.Stage;

import org.jetbrains.annotations.NotNull;

import etu.leti.parser.MapParser;
import org.w3c.dom.Text;

public class Controller implements Initializable {

    enum Mode {
        STEP_BY_STEP, JUST_RESULT, NO_MODE
    }

    private static final int fieldWidth = 20;
    private static final int fieldHeight = 21;

    private MapParser mapParser;
    private FieldVisualizer fieldVisualizer;

    Stage primaryStage;

    private Mode currentMode = Mode.NO_MODE;

    // For save/load file
    private Alert alert;

    @FXML
    private VBox mainStage;
    @FXML
    private GridPane mainVisualField;
    @FXML
    private Button runButton;
    @FXML
    private Button stepButton;
    @FXML
    private ChoiceBox<String> modeChoiceBox;
    @FXML
    private TextArea logTextField;
    @FXML
    private TextArea algAboutTextArea;

    // MENU BAR ITEMS
```

```
    @FXML
    private MenuItem saveButton;
    @FXML
    private MenuItem loadButton;
    @FXML
    private MenuItem resetButton;
    @FXML
    private MenuItem generateMapButton;
    @FXML
    private MenuItem aboutAlgButton;
    @FXML
    private MenuItem aboutButton;

    @Override
    public void initialize(URL location, ResourceBundle resources)
{
        loadDataForChoiceBox();
        initAlert();

        modeChoiceBox.setOnAction(this::modeChanged);
        mapParser = new MapParser();
        fieldVisualizer = new FieldVisualizer(mainVisualField,
this.getClass().getClassLoader(), fieldWidth, fieldHeight);
    }

    public void runAlgorithm(ActionEvent event) {
        if(currentMode == Mode.NO_MODE) {
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Mode alert");
            alert.setHeaderText("Selected mode");
            alert.setContentText("You didn't select the mode in
which the algorithm should work. " +
                                 "Please do this and try again.");

            alert.showAndWait();
        } else if(currentMode == Mode.JUST_RESULT) {
            fieldVisualizer.showResultWay();
        } else {
            fieldVisualizer.stepByStepWay();
            runButton.setDisable(true);
        }
    }

    public void madeOneStep(ActionEvent event) {
        if(fieldVisualizer.makeOneStep()) {
            runButton.setDisable(false);
        }
    }

    public void loadFile(@NotNull ActionEvent event) throws
IOException {
        final FileChooser fileChooser = new FileChooser();
```

```
        fileChooser.getExtensionFilters().addAll((new
FileChooser.ExtensionFilter("JSON", "*.json")));
        File file =
fileChooser.showOpenDialog(mainStage.getScene().getWindow());

        mapParser.setJsonFile(file);
        Cell[] cellFieldFromFile;

        try {
            cellFieldFromFile = mapParser.parseCellArr();
        } catch (FileNotFoundException exc) {
            alert.setContentText("No valid file with map was
choosen!");
            alert.showAndWait();
            return;
        } catch (JsonParseException exc) {
            alert.setContentText("JSON file data is incorrect!");
            alert.showAndWait();
            return;
        }

        fieldVisualizer.setWorkingMap(cellFieldFromFile);

        try {
            fieldVisualizer.fillGridByCell();
        } catch (JsonParseException |
ArrayIndexOutOfBoundsException exc) {
            alert.setContentText(exc.getMessage());
            alert.showAndWait();
        }
    }

    public void saveFile(ActionEvent event) throws IOException {
        final FileChooser fileChooser = new FileChooser();
        fileChooser.getExtensionFilters().addAll((new
FileChooser.ExtensionFilter("JSON", "*.json")));
        File file =
fileChooser.showSaveDialog(mainStage.getScene().getWindow());

        mapParser.setJsonFile(file);

        if(file == null) {
            alert.setContentText("No valid file was choosen");
            alert.showAndWait();
            return;
        }


mapParser.writeCellArray(fieldVisualizer.getWorkingMapAsArray());
    }

    public void resetField(ActionEvent event) {
```

```
        fieldVisualizer.resetField();
    }

    public void generateMap(ActionEvent event) {
        fieldVisualizer.resetField();
        fieldVisualizer.createNewMap();
    }

    public void getAlgInformation(ActionEvent event) throws
IOException {
        ExtraWindowShower.showProgInfoWindow(primaryStage,
getClass().getClassLoader());
    }

    public void getAuthorInformation(ActionEvent event) {
        ExtraWindowShower.showAboutWindow(primaryStage);
    }

    public void setStage(Stage primaryStage){
        this.primaryStage = primaryStage;
    }

    private void loadDataForChoiceBox() {
        // For ChoiceBox variants
        ObservableList<String> listOfModesNames =
FXCollections.observableArrayList();
        // delete all possible data for avoid duplication
        listOfModesNames.removeAll();
        String stepByStepStr = "Step by step";
        String justResultStr = "Just result";
        listOfModesNames.addAll(stepByStepStr, justResultStr);
        modeChoiceBox.getItems().addAll(listOfModesNames);
    }

    private void initAlert() {
        alert = new Alert(Alert.AlertType.WARNING);
        alert.setTitle("Warning alert");
        alert.setHeaderText("JSON map file");
    }

    @FXML
    private void modeChanged(ActionEvent event) {
        if(modeChoiceBox.getValue().equals("Step by step")) {
            currentMode = Mode.STEP_BY_STEP;
        } else if(modeChoiceBox.getValue().equals("Just result"))
{
            runButton.setDisable(false);
            currentMode = Mode.JUST_RESULT;
        }
    }
}
```

Файл GridWorker.java

```
package etu.leti.gui.gridhandler;

import com.google.gson.JsonParseException;
import etu.leti.field.Cell;
import javafx.scene.Node;
import javafx.scene.image.Image;
import javafx.scene.layout.GridPane;
import org.jetbrains.annotations.NotNull;
import org.jetbrains.annotations.Nullable;

import java.util.Objects;

public class GridWorker {

    private final GridPane mainVisualField;
    private ImageCell[][] fillGridImages;
    private ImageCell playerImg;
    private final ClassLoader classLoader;
    private ImageCell lastImage;

    public GridWorker(GridPane mainVisualField, @NotNull
ClassLoader classLoader) {
        this.mainVisualField = mainVisualField;
        this.classLoader = classLoader;
        lastImage = null;
    }

    /**
     * Get a specific node from GridPane by is column and row
index
     * @param col
     * Column index
     * @param row
     * Row index
     * @return
     * Node with specified coordinates
     */
    public @Nullable Node getNodeFromGridPane(int col, int row) {
        for (Node node : mainVisualField.getChildren()) {
            if (GridPane.getColumnIndex(node) == col &&
GridPane.getRowIndex(node) == row) {
                return node;
            }
        }
        return null;
    }

    public void fillGridByCell(Cell[] @NotNull [] cells) {
```

```
        playerImg = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
player.png"))));
        int x, y, i = 0;
        fillGridImages = new ImageCell[cells.length]
[cells[0].length];

        for(Cell[] cellArr : cells) {
            for (Cell cell : cellArr) {
                if (cell == null) {
                    break;
                }

                x = cell.getPosX();
                y = cell.getPosY();
                switch (cell.getOreInCellType()) {
                    case GROUND:
                        fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
ground.jpg"))));
                        break;
                    case GOLD_ORE:
                        fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
gold.jpg"))));
                        break;
                    case IRON_ORE:
                        fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
iron.jpg"))));
                        break;
                    case STONE_ORE:
                        fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
stone.jpg"))));
                        break;
                    case HELL:
                        fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
hell.jpeg"))));
                        break;
                    case AIR:
                        fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
air.png"))));
                }
                mainVisualField.add(fillGridImages[x][y], x, y);

                ++i;
            }
        }
```

```
        if(i == cells.length * cells[0].length) {
            mainVisualField.add(playerImg, 0, 0);
            return;
        }

        cleanGridCells();
        throw new JsonParseException("Incorrect data for " + i + "
element in Cell array from file");
    }

    public void deleteCharacter() {
        mainVisualField.getChildren().remove(playerImg);
    }

    public void showOneStep(Cell @NotNull [] shortestWay, int
shortestWayCurrPos) {
        if(shortestWayCurrPos == shortestWay.length - 1) {
            return;
        }

        if(playerImg != null) {
            mainVisualField.getChildren().remove(playerImg);
            playerImg = null;
        }

        int x = shortestWay[shortestWayCurrPos].getPosX();
        int y = shortestWay[shortestWayCurrPos].getPosY();
        int extraX, extraY;

        if(lastImage != null) {
            extraX = shortestWay[shortestWayCurrPos +
1].getPosX();
            extraY = shortestWay[shortestWayCurrPos +
1].getPosY();
            fillGridImages[extraX][extraY] = lastImage;
            mainVisualField.add(fillGridImages[extraX][extraY],
extraX, extraY);
        }

        lastImage = fillGridImages[x][y];
        mainVisualField.getChildren().remove(fillGridImages[x]
[y]);

        switch
(shortestWay[shortestWayCurrPos].getOreInCellType()) {
            case GROUND:
                fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
move/ground.png"))));
                break;
            case GOLD_ORE:
```

```
                fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
move/gold.png"))));
                break;
            case IRON_ORE:
                fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
move/iron.png"))));
                break;
            case STONE_ORE:
                fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
move/stone.png"))));
                break;
            case HELL:
                fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
move/hell.png"))));
                break;
            case AIR:
                fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
move/air.png"))));
        }
        mainVisualField.add(fillGridImages[x][y], x, y);
    }

    public void visualizeShortestWay(Cell @NotNull []
shortWayCells) {
        int x, y;

        for(Cell cell : shortWayCells) {
            x = cell.getPosX();
            y = cell.getPosY();
            if (x == 0 & y == 0) {
                continue;
            }
            mainVisualField.getChildren().remove(fillGridImages[x]
[y]);

            switch (cell.getOreInCellType()) {
                case GROUND:
                    fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
choose/ground.png"))));
                    break;
                case GOLD_ORE:
                    fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
choose/gold.png"))));
                    break;
                case IRON_ORE:
```

```
                    fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
choose/iron.png"))));
                    break;
                case STONE_ORE:
                    fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
choose/stone.png"))));
                    break;
                case AIR:
                    fillGridImages[x][y] = new ImageCell(new
Image(Objects.requireNonNull(classLoader.getResourceAsStream("img/
choose/air.png"))));
            }
            mainVisualField.add(fillGridImages[x][y], x, y);
        }
    }

    public void cleanGridCells() {
//        if(fillGridImages == null) {
//            return;
//        } else {
//            for(ImageCell[] cellArr : fillGridImages) {
//                for (ImageCell cell : cellArr) {
//                    if(cell != null) {
//
mainVisualField.getChildren().remove(cell);
//                    }
//                }
//            }
//        }

mainVisualField.getChildren().retainAll(mainVisualField.getChildre
n().get(0));
        if(playerImg != null) {
            mainVisualField.getChildren().remove(playerImg);
        }
        if(lastImage != null) {
            mainVisualField.getChildren().remove(lastImage);
            lastImage = null;
        }
    }
}
```

Файл ImageCell.java

```
package etu.leti.gui.gridhandler;

import javafx.beans.property.ObjectProperty;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.geometry.VPos;
```

```
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.Region;

/**
 * Class for implementation of fitting image insertion into
GridPane
 */
public class ImageCell extends Region {

    private final ImageView imageView;

    public ImageCell(Image image) {
        imageView = new ImageView(image);
        getChildren().add(imageView);
    }

    public ImageCell() {
        this(null);
    }

    public final void setImage(Image value) {
        imageView.setImage(value);
    }

    public final Image getImage() {
        return imageView.getImage();
    }

    public final ObjectProperty<Image> imageProperty() {
        return imageView.imageProperty();
    }

    @Override
    protected void layoutChildren() {
        super.layoutChildren();
        Insets insets = getInsets();
        double x = insets.getLeft();
        double y = insets.getTop();
        double width = getWidth() - x - insets.getRight();
        double height = getHeight() - y - insets.getBottom();

        Image image = getImage();
        double imageWidth = 0;
        double imageHeight = 0;
        if (image != null) {
            imageWidth = image.getWidth();
            imageHeight = image.getHeight();
        }

        // scale ImageView to available size
```

```
        double factor = Math.min(width / imageWidth, height /
imageHeight);
        if (Double.isFinite(factor) && factor > 0) {
            imageView.setFitHeight(factor * imageHeight);
            imageView.setFitWidth(factor * imageWidth);
            imageView.setVisible(true);
        } else {
            imageView.setVisible(false);
        }

        // center ImageView in available area
        layoutInArea(imageView, x, y, width, height, 0,
HPos.CENTER, VPos.CENTER);
    }

}
```

Файл MapParser.java

```
package etu.leti.parser;

import com.google.gson.Gson;

import java.io.*;

import com.google.gson.GsonBuilder;
import com.google.gson.JsonParseException;
import etu.leti.field.*;
import org.jetbrains.annotations.NotNull;

public class MapParser {

    private File jsonFile;
    private final Gson gson;

    public MapParser() {
        gson = new GsonBuilder()
                .setPrettyPrinting()
                .excludeFieldsWithoutExposeAnnotation()
                .create();
    }

    public File getJsonFile() {
        return jsonFile;
    }

    public void setJsonFile(File jsonFile) {
        this.jsonFile = jsonFile;
    }

    public Cell[] parseCellArr() throws IOException {
```

```
        if(jsonFile == null) {
            throw new FileNotFoundException("No json file was
choose");
        }

        BufferedReader reader = new BufferedReader(new
FileReader(jsonFile));
        StringBuilder userJson = new StringBuilder();
        String line;
        while ((line = reader.readLine()) != null) {
            userJson.append(line);
        }
        reader.close();

        Cell[] cells = gson.fromJson(userJson.toString(),
Cell[].class);

        if(cells == null) {
            throw new JsonParseException("Empty JSON map file");
        }

        initFileNames(cells);

        return cells;
    }

    public void writeCellArray(Cell[] cellArray) throws
IOException {

        if(jsonFile == null) {
            throw new FileNotFoundException("No json file was
choose");
        }

        if(jsonFile.exists()) {
            jsonFile.delete();
        }
        jsonFile.createNewFile();

        String jsonStr = gson.toJson(cellArray);

        BufferedWriter writer = new BufferedWriter(new
FileWriter(jsonFile));
        writer.write(jsonStr);
        writer.close();
    }

    public static void initFileNames(Cell @NotNull [] cells) {
        int i = 0;
        for(Cell cell : cells) {
            if(cell == null) {
```

```
                throw new JsonParseException("Incorrect data for "
+ i + " element in Cell array from file");
            }
            switch (cell.getOreInCellType()) {
                case GROUND:
                    cell.setOreImg("img/ground.jpg");
                    break;
                case GOLD_ORE:
                    cell.setOreImg("img/gold.jpg");
                    break;
                case IRON_ORE:
                    cell.setOreImg("img/iron.jpg");
                    break;
                case STONE_ORE:
                    cell.setOreImg("img/stone.jpg");
                    break;
                case HELL:
                    cell.setOreImg("img/hell.jpeg");
                    break;
                default:
                    cell.setOreImg("img/air.png");
            }
            i++;
        }
    }
}
```

Файл Graph.java

```
package etu.leti.algorithm;

import etu.leti.field.Cell;
import etu.leti.field.Field;
import etu.leti.ore.OreTypes;
import org.jetbrains.annotations.NotNull;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.PriorityQueue;

public class Graph {
    private Field field;
    private Integer hell;
    private Cell hellCell;
    private HashMap<Integer, ArrayList<Cell>> veins;
    private Integer start;
    private Cell startCell;
    private HashMap<Integer, ArrayList<Node>> graph;
    private double lengthMinWay;
    private ArrayList<Node> nodesOfShortestWay;
    private ArrayList<Cell> cellsOfShortestWay;
```

```
    public Graph(Field field) {
        this.field = field;
        veins = new HashMap<>();
        graph = new HashMap<>();
        nodesOfShortestWay = new ArrayList<>();
        cellsOfShortestWay = new ArrayList<>();
    }

    /**
     * Method, which find all cells belong to one vein
     *
     * @param x    - pos X of cell
     * @param y    - pos Y of cell
     * @param type - type Ore in the cell
     * @param vein - ArrayList, where storage all cells belong to
one vein
     */
    private void findVein(int x, int y, OreTypes type,
ArrayList<Cell> vein) {
        Cell cell = field.getCells()[x][y];
        if (!cell.isChecked()) {
            cell.setChecked(true);
            vein.add(cell);
        }
        if (cell.getPosX() + 1 < field.getHeight()) {
            Cell downCell = field.getCells()[cell.getPosX() + 1]
[cell.getPosY()];
            if (!downCell.isChecked() &&
downCell.getOre().getOreType() == type) {
                findVein(cell.getPosX() + 1, cell.getPosY(), type,
vein);
            }
        }
        if (cell.getPosY() + 1 < field.getWidth()) {
            Cell rightCell = field.getCells()[cell.getPosX()]
[cell.getPosY() + 1];
            if (!rightCell.isChecked() &&
rightCell.getOre().getOreType() == type) {
                findVein(cell.getPosX(), cell.getPosY() + 1, type,
vein);
            }
        }
        if (cell.getPosY() - 1 >= 0) {
            Cell leftCell = field.getCells()[cell.getPosX()]
[cell.getPosY() - 1];
            if (!leftCell.isChecked() &&
leftCell.getOre().getOreType() == type) {
                findVein(cell.getPosX(), cell.getPosY() - 1, type,
vein);
            }
        }
```

```
        if (cell.getPosX() - 1 >= 0) {
            Cell upCell = field.getCells()[cell.getPosX() - 1]
[cell.getPosY()];
            if (!upCell.isChecked() &&
upCell.getOre().getOreType() == type) {
                findVein(cell.getPosX() - 1, cell.getPosY(), type,
vein);
            }
        }
    }

    /**
     * Method, which find all veins and add its to
HashMap<Integer, ArrayList<Cell>> veins
     */

    private void findVeins() {
        Integer i = 0;
        for (Cell[] horizontalCells : field.getCells()) {
            for (Cell cell : horizontalCells) {
                if(cell.getPosX() == 0 && cell.getPosY() == 0 &&
cell.getOre().getOreType() == OreTypes.AIR){
                    ArrayList<Cell> vein = new ArrayList<>();
                    this.start = i;
                    this.startCell = cell;
                    vein.add(cell);
                    veins.put(i, vein);
                    i++;
                    continue;
                }
                if (cell.isChecked()) {
                    continue;
                }
                if (cell.getOre().getOreType() ==
OreTypes.GOLD_ORE ||
                        cell.getOre().getOreType() ==
OreTypes.IRON_ORE ||
                        cell.getOre().getOreType() ==
OreTypes.STONE_ORE ||
                        cell.getOre().getOreType() ==
OreTypes.HELL) {

                    ArrayList<Cell> vein = new ArrayList<>();
                    if (cell.getOre().getOreType() ==
OreTypes.HELL) {
                        this.hell = i;
                        this.hellCell = cell;
                        vein.add(cell);
                        veins.put(i, vein);
                        i++;
                        continue;
                    }
```

```
                    findVein(cell.getPosX(), cell.getPosY(),
cell.getOre().getOreType(), vein);
                    veins.put(i, vein);
                    i++;

                }
            }
        }
    }

    private void setNodesToGraph(Integer numberVein1, Integer
numberVein2, Cell fromVein1, Cell toVein2, double minWay){
        if(graph.get(numberVein1) == null){
            ArrayList<Node> nodes = new ArrayList<>();
            nodes.add(new Node(numberVein2, fromVein1, toVein2,
minWay));
            graph.put(numberVein1, nodes);
        }else{
            graph.get(numberVein1).add(new Node(numberVein2,
fromVein1, toVein2, minWay));
        }
    }

    /**
     * Method, which find two coming cells between vein1 and vein
two
     *
     * @param vein1       - ArrayList<Cell> of vein, from which
the path is sought
     * @param vein2       - ArrayList<Cell> of vein, to which the
path is sought
     * @param numberVein1 - Integer number of vein, from which the
path is sought
     * @param numberVein2 - Integer number of vein, to which the
path is sought
     */
    private void findMinWayBetweenTwoVeins(@NotNull
ArrayList<Cell> vein1, ArrayList<Cell> vein2, Integer numberVein1,
Integer numberVein2) {
        double minWay = Double.MAX_VALUE;
        Cell fromVein1 = new Cell();
        Cell toVein2 = new Cell();
        for (Cell cell1 : vein1) {
            for (Cell cell2 : vein2) {
                double dist = Math.abs(cell1.getPosX() -
cell2.getPosX()) + Math.abs(cell1.getPosY() - cell2.getPosY()) -
1;
                if (minWay > dist) {
                    fromVein1 = cell1;
                    toVein2 = cell2;
                    minWay = dist;
                }
```

```
            }
        }
        setNodesToGraph(numberVein1, numberVein2, fromVein1,
toVein2, minWay);
        setNodesToGraph(numberVein2, numberVein1, toVein2,
fromVein1, minWay);
    }

    /**
     * Method, which find min way between all veins. Way - two
coming cells
     */
    private void findMinWayBetweenVeins() {
        findVeins();
        int k = 1;
        for (HashMap.Entry<Integer, ArrayList<Cell>> entry :
veins.entrySet()) {
            for (int i = k; i < veins.size(); i++) {
                findMinWayBetweenTwoVeins(entry.getValue(),
veins.get(i), entry.getKey(), i);
            }
            k++;
        }
    }

    private void dijkstra(){
        findMinWayBetweenVeins();
        PriorityQueue<Node> vertexQueue = new PriorityQueue<>();
        vertexQueue.add(new Node(start, startCell));
        while (!vertexQueue.isEmpty()){
            Node vertex = vertexQueue.poll();
            for(Node node : graph.get(vertex.getVertex())){
                double weight = node.getWeight();
                double distance = vertex.getMinDistance() +
weight;
                if(distance < node.getMinDistance()){
                    for(Node buf : graph.get(node.getVertex())){

if(buf.getVertex().equals(vertex.getVertex())){
                            buf.setMinDistance(distance);
                            buf.setPrev(vertex);
                            break;
                        }
                    }
                    node.setMinDistance(distance);
                    node.setPrev(vertex);
                    vertexQueue.add(node);
                }
            }
        }
    }
```

```
    private void setCellsOfGround(@NotNull Cell from, @NotNull
Cell to){
        int x;
        int y;
        int dist;
        int startPos;
        if(from.getPosY() == to.getPosY()){
            y = to.getPosY();
            dist = Math.abs(from.getPosX() - to.getPosX());
            startPos = Math.min(from.getPosX(), to.getPosX());
            for(int i = 1; i < dist; i++){
                cellsOfShortestWay.add(field.getCells()[+
+startPos][y]);
            }
        }else if(from.getPosX() == to.getPosX()){
            x = to.getPosX();
            dist = Math.abs(from.getPosY() - to.getPosY());
            startPos = Math.min(from.getPosY(), to.getPosY());
            for(int i = 1; i < dist; i++){
                cellsOfShortestWay.add(field.getCells()[x][+
+startPos]);
            }
        }else {
            int distY = Math.abs(from.getPosY() - to.getPosY()) -
1;
            int distX = Math.abs(from.getPosX() - to.getPosX());
            int startPosY;
            int startPosX;
            if(from.getPosX() < to.getPosX()){
                startPosY = from.getPosY();
                startPosX = from.getPosX();
            }else{
                startPosY = to.getPosY();
                startPosX = to.getPosX();
            }
            for(int i = 0; i < distX; i++){
                cellsOfShortestWay.add(field.getCells()[+
+startPosX][startPosY]);
            }
            startPosY = Math.min(from.getPosY(), to.getPosY());
            for (int i = 0; i < distY; i++){
                cellsOfShortestWay.add(field.getCells()[startPosX]
[++startPosY]);
            }
        }
    }

    private void shortestWay(){
        dijkstra();

        double minDist = Double.MAX_VALUE;
```

```
        Node minNode = new Node();
        for(Node node : graph.get(hell)){
            if(minDist > node.getMinDistance()){
                minDist = node.getMinDistance();
                minNode = node;
            }
        }

        lengthMinWay = minDist;

        nodesOfShortestWay.add(new Node(hell, minNode.getTo(),
hellCell));

        while (minNode.getPrev() != null){
            nodesOfShortestWay.add(minNode.getPrev());
            minNode = minNode.getPrev();
        }

        Collections.reverse(nodesOfShortestWay);

        for(Node node : nodesOfShortestWay){
            if(!node.getVertex().equals(start)){
                cellsOfShortestWay.add(node.getFrom());
                setCellsOfGround(node.getFrom(), node.getTo());
                cellsOfShortestWay.add(node.getTo());
            }

        }
    }

    public HashMap<Integer, ArrayList<Cell>> getVeins() {
        return veins;
    }

    public HashMap<Integer, ArrayList<Node>> getGraph() {
        return graph;
    }

    public double getLengthMinWay() {
        return lengthMinWay;
    }

    public ArrayList<Cell> getCellsOfShortestWay() {
        shortestWay();
        return cellsOfShortestWay;
    }

    public Cell getHellCell() {
        return hellCell;
    }
}
```

Файл Node.java

```
package etu.leti.algorithm;

import etu.leti.field.Cell;
import org.jetbrains.annotations.NotNull;

public class Node implements Comparable<Node>{
    private Integer vertex;
    private Cell from;
    private Cell to;
    private double weight;
    private double minDistance;
    private Node prev;

    public Node() {
    }

    public Node(Integer vertex, Cell from){
        this.vertex = vertex;
        this.from = from;
        this.minDistance = 0;
    }

    public Node(Integer vertex, Cell from, Cell to) {
        this.vertex = vertex;
        this.from = from;
        this.to = to;
        this.minDistance = Double.MAX_VALUE;
    }

    public Node(Integer vertex, Cell from, Cell to, double weight)
{
        this.vertex = vertex;
        this.from = from;
        this.to = to;
        this.weight = weight;
        this.minDistance = Double.MAX_VALUE;
    }

    public Integer getVertex() {
        return vertex;
    }

    public double getWeight() {
        return weight;
    }

    public double getMinDistance() {
        return minDistance;
    }
```

```
    public void setMinDistance(double minDistance) {
        this.minDistance = minDistance;
    }

    @Override
    public int compareTo(@NotNull Node other) {
        return Double.compare(minDistance, other.minDistance);
    }

    public Node getPrev() {
        return prev;
    }

    public void setPrev(Node prev) {
        this.prev = prev;
    }

    public Cell getFrom() {
        return from;
    }

    public Cell getTo() {
        return to;
    }
}
```

Файл Cell.java

```
package etu.leti.field;

import com.google.gson.annotations.Expose;
import etu.leti.ore.Ore;
import etu.leti.ore.OreTypes;
import org.jetbrains.annotations.NotNull;

public class Cell implements Comparable<Cell> {
    @Expose
    private int posX;
    @Expose
    private int posY;
    @Expose(serialize = false, deserialize = true)
    private boolean checked;
    @Expose
    private Ore ore;

    public Cell() { }

    public Cell(@NotNull Cell cell) {
        this.posX = cell.posX;
        this.posY = cell.posY;
        this.checked = cell.checked;
        this.ore = cell.ore;
```

```
}

public Cell(int posX, int posY, Ore ore) {
    this.posX = posX;
    this.posY = posY;
    this.ore = ore;
}

public int compareTo(@NotNull Cell cell) {
    if(this.posX == cell.posX & this.posY == cell.posY) {
        return 0;
    } else if(this.posX < cell.posX & this.posY < cell.posY) {
        return 1;
    } else if(this.posX > cell.posX & this.posY > cell.posY) {
        return -1;
    } else if(this.posY < cell.posY) {
        return 1;
    } else {
        return -1;
    }
}

public int getPosX() {
    return posX;
}

public void setPosX(int posX) {
    this.posX = posX;
}

public int getPosY() {
    return posY;
}

public void setPosY(int posY) {
    this.posY = posY;
}

public Ore getOre() {
    return ore;
}

public void setOre(Ore ore) {
    this.ore = ore;
}

public boolean isChecked() {
    return checked;
}

public void setChecked(boolean checked) {
    this.checked = checked;
```

```
    }

    public OreTypes getOreInCellType() {
        return ore.getOreType();
    }

    public void setOreImg(String fileName) {
        ore.setPath(fileName);
    }

    public void swapCoords() {
        int temp = posX;
        posX = posY;
        posY = temp;
    }
}
```

Файл Field.java

```
package etu.leti.field;

public class Field {
    private Cell[][] cells;
    private int height;
    private int width;

    public Field() {
    }

    public Field(Cell[][] cells, int height, int width) {
        this.cells = cells;
        this.height = height;
        this.width = width;
    }

    public Cell[][] getCells() {
        return cells;
    }

    public void setCells(Cell[][] cells) {
        this.cells = cells;
    }

    public int getHeight() {
        return height;
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public int getWidth() {
```

```
        return width;
    }

    public void setWidth(int width) {
        this.width = width;
    }
}
```

Файл MapGenerator.java

```
package etu.leti.generator;

import etu.leti.field.Cell;
import java.util.Random;
import etu.leti.ore.Ore;
import etu.leti.ore.OreTypes;
import java.util.HashMap;

public class MapGenerator {
    int currX = 0, currY = 0, randX = 0, randY = 0;
    int randOre = 0, randSize = 0, countVeinsInSector = 0, rand =
0;
    int veinX = 0, veinY = 0;
    int x = 0 , y = 0;

    Cell[][] field;
    HashMap<Integer, OreTypes> ore;
    HashMap<Integer, String> picture;
    Random random;

    public MapGenerator(int initX, int initY) {
       random = new Random();
       x = initX;
       y = initY;
       field = new Cell[x][y];
       initOreList();
    }

    private void initOreList() {
        ore = new HashMap<>();
        ore.put(1, OreTypes.GOLD_ORE);
        ore.put(2, OreTypes.IRON_ORE);
        ore.put(3, OreTypes.STONE_ORE);
        ore.put(4, OreTypes.HELL);
        ore.put(10, OreTypes.GROUND);
        ore.put(11, OreTypes.AIR);

        picture = new HashMap<>();
        picture.put(1, "img/gold.jpg");
        picture.put(2, "img/iron.jpg");
        picture.put(3, "img/stone.jpg");
        picture.put(4, "img/hell.jpeg");
```

```
        picture.put(10, "img/ground.jpg");
        picture.put(11, "air.jpg");
    }

    private void changeQuarter(int arg, int x, int y) {
        if (arg == 0)
        { currX = 0; currY = 0;}
        if (arg == 1)
        { currX = x / 2; currY = 0;}
        if (arg == 2)
        { currX = 0; currY = y / 2;}
        if (arg == 3)
        { currX = x / 2; currY = y / 2;}
    }

    private void randomizeOre(int arg, int x, int y) {

        randX = random.nextInt((x / 2)) + currX;
        randY = random.nextInt((x / 2)) + currY;
        if (arg == 1) {
            randOre = random.nextInt(3) + 1;
            randSize = random.nextInt(4) + random.nextInt(4) + 2;
        }
    }

    private void randomizeVeinsCount() {
        rand = random.nextInt(100 ) + 1;
        if (rand > 60) {
            countVeinsInSector = 3;
        }
        if (rand > 10) {
            countVeinsInSector = 2;
        } else {
            countVeinsInSector = 1;
        }
    }


    private void dirtAndAirFiller(int x, int y) {
        for (int i = 0; i < 1; ++i) {
            for (int j = 0; j < x; ++j){
                field[j][i] = new Cell(j, i, new
Ore(picture.get(11), ore.get(11)));
            }
        }
        for (int i = 1; i < y; ++i) {
            for (int j = 0; j < x; ++j) {
                if (field[j][i] == null) {
                    field[j][i] = new Cell(j, i, new
Ore(picture.get(10), ore.get(10)));
                }
            }
```

```
        }
    }

    public void reset() {
        currX = 0; currY = 0; randX = 0; randY = 0;
        randOre = 0; randSize = 0; countVeinsInSector = 0; rand =
0;
        veinX = 0; veinY = 0;
        field = new Cell[x][y];
    }

    private int checkNeighbors(int xC, int yC, int prevX, int
PrevY) {
       for(int i = yC - 1; i < yC + 1; ++i) {
           for(int j = xC - 1;j < xC + 1; ++j) {
               if (i > 0 && j > 0 && j < x && i < y) {
                   if(field[j][i]!=null ) {
                       if (field[j][i].getOre().getOreType()!
=field[prevX][PrevY].getOre().getOreType()) {
                           return 0;
                       }
                   }
               }
           }
       }
       return 1;
    }

    public Cell[][] generateMap() {
        rand = random.nextInt(x);
        field[rand][y-1] = new Cell(rand, y-1, new
Ore(picture.get(4), ore.get(4)));

        for (int i = 0; i < 4; ++i) {
            randomizeVeinsCount();
            changeQuarter(i, x, y);
            for (int n = 0; n < countVeinsInSector; ++n) {
                randomizeOre(1, x, y);
                for(int g = 0; g < 10; ++g) {
                    if (field[randX][randY] == null) {
                        g = 10;
                        field[randX][randY] = new Cell(randX,
randY, new Ore(picture.get(randOre), ore.get(randOre)));
                        veinX = randX;
                        veinY = randY;
                        for (int k = 0; k < randSize; ++k) {
                            randX = random.nextInt(3) - 1;
                            if (randX == 0) {
                                randY = random.nextInt(3) - 1;
                                if (randX == 0 && randY == 0){
                                    k--;
                                }
```

```
                                }
                                else
                                    randY = 0;
                                if (veinX + randX < x && veinY + randY
<y && veinX + randX > 0 && veinY + randY > 0) {
                                    if (field[veinX + randX][veinY +
randY] == null && checkNeighbors(veinX+ randX,veinY +
randY,veinX,veinY)==1) {
                                        veinX += randX;
                                        veinY += randY;
                                        field[veinX][veinY] = new
Cell(veinX, veinY, new Ore(picture.get(randOre),
ore.get(randOre)));
                                    }
                                }
                            }
                        } else {
                            randomizeOre(0, x, y);
                        }
                    }
                }

            }
            dirtAndAirFiller(x, y);
            return field;
        }
}
```

Файл Ore.java

```
package etu.leti.ore;

import com.google.gson.annotations.Expose;
import org.jetbrains.annotations.NotNull;

public class Ore {
    @Expose(serialize = false, deserialize = true)
    protected String imgFileName;
    @Expose
    protected OreTypes oreType;

    public Ore() {}

    public Ore(@NotNull Ore ore) {
        this.imgFileName = ore.imgFileName;
        this.oreType = ore.oreType;
    }

    public Ore(OreTypes oreType){
        this.oreType = oreType;
    }
```

```
    public Ore(String imgFileName, OreTypes oreType) {
        this.imgFileName = imgFileName;
        this.oreType = oreType;
    }

    public String getPath() {
        return imgFileName;
    }

    public void setPath(String imgFileName) {
        this.imgFileName = imgFileName;
    }

    public OreTypes getOreType() {
        return oreType;
    }

    public void setOreType(OreTypes oreType) {
        this.oreType = oreType;
    }
}
```

Файл OreTypes.java

```
package etu.leti.ore;

public enum OreTypes {
    GROUND,
    GOLD_ORE,
    IRON_ORE,
    STONE_ORE,
    AIR,
    HELL
}
```

Файл algorithm.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.TextArea?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<AnchorPane maxHeight="400.0" maxWidth="600.0" minHeight="215.0"
minWidth="600.0" prefHeight="215.0" prefWidth="600.0"
xmlns="http://javafx.com/javafx/11.0.1"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="etu.leti.gui.ExtraWindowShower">
```

```
   <Text layoutX="14.0" layoutY="29.0" strokeType="OUTSIDE"
strokeWidth="0.0" text="About program and algorithm">
      <font>
         <Font name="System Bold" size="16.0" />
      </font>
   </Text>
   <VBox layoutX="354.0" layoutY="110.0" prefHeight="200.0"
prefWidth="100.0" />
   <TextArea fx:id="algAboutTextArea" editable="false"
layoutX="14.0" layoutY="33.0" prefHeight="176.0" prefWidth="570.0"
text="This program uses Dijkstra's algorithm, which works on a
graph created from &#10;stranded cores. The weight of the edges is
formed by blocks of earth &#10;(the transition inside the veins is
free).&#10;&#10;&#10;The program can display the completed path in
both step-by-step &#10;and &quot;All at once&quot;
mode.&#10;&#10;You can also use the appropriate commands to
generate &#10;new maps or use saved ones." />
</AnchorPane>
```

Файл miner.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.ChoiceBox?>
<?import javafx.scene.control.Menu?>
<?import javafx.scene.control.MenuBar?>
<?import javafx.scene.control.MenuItem?>
<?import javafx.scene.control.TextArea?>
<?import javafx.scene.layout.ColumnConstraints?>
<?import javafx.scene.layout.GridPane?>
<?import javafx.scene.layout.HBox?>
<?import javafx.scene.layout.RowConstraints?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<VBox fx:id="mainStage" maxHeight="-Infinity" maxWidth="-Infinity"
minHeight="-Infinity" minWidth="-Infinity" prefHeight="768.0"
prefWidth="1024.0" xmlns="http://javafx.com/javafx/11.0.1"
xmlns:fx="http://javafx.com/fxml/1"
fx:controller="etu.leti.gui.Controller">
    <MenuBar prefWidth="871.0">
        <Menu mnemonicParsing="false" text="File">
            <MenuItem fx:id="saveButton" mnemonicParsing="false"
onAction="#saveFile" text="Save" />
            <MenuItem fx:id="loadButton" mnemonicParsing="false"
onAction="#loadFile" text="Load" />
        </Menu>
        <Menu mnemonicParsing="false" text="Edit">
```

```
            <MenuItem fx:id="resetButton" mnemonicParsing="false"
onAction="#resetField" text="Reset" />
         <MenuItem fx:id="generateMapButton"
mnemonicParsing="false" onAction="#generateMap" text="Generate" />
        </Menu>
        <Menu mnemonicParsing="false" text="Help">
            <MenuItem fx:id="aboutAlgButton"
mnemonicParsing="false" onAction="#getAlgInformation"
text="Algorithm" />
            <MenuItem fx:id="aboutButton" mnemonicParsing="false"
onAction="#getAuthorInformation" text="About" />
        </Menu>
    </MenuBar>
    <HBox prefHeight="1024.0" prefWidth="1280.0">
        <GridPane fx:id="mainVisualField" alignment="CENTER"
gridLinesVisible="true" maxHeight="740.0" maxWidth="705.0"
minHeight="740.0" minWidth="705.0" prefHeight="740.0"
prefWidth="705.0" scaleShape="false">
            <columnConstraints>
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
```

```
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
                <ColumnConstraints hgrow="SOMETIMES"
minWidth="10.0" prefWidth="100.0" />
            </columnConstraints>
            <rowConstraints>
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
                <RowConstraints minHeight="10.0" prefHeight="30.0"
vgrow="SOMETIMES" />
```

```
            </rowConstraints>
        </GridPane>
        <VBox alignment="BOTTOM_CENTER" prefHeight="740.0"
prefWidth="319.0">
            <Button fx:id="runButton" mnemonicParsing="false"
onAction="#runAlgorithm" prefHeight="57.0"
                    prefWidth="125.0" text="Run">
                <VBox.margin>
                    <Insets bottom="110.0"/>
                </VBox.margin>
                <font>
                    <Font size="16.0"/>
                </font>
            </Button>
            <Text strokeType="OUTSIDE" strokeWidth="0.0" text="Run
mode">
                <font>
                    <Font size="16.0"/>
                </font>
            </Text>
            <ChoiceBox fx:id="modeChoiceBox" prefWidth="150.0"
onAction="#modeChanged">
                <VBox.margin>
                    <Insets bottom="60.0"/>
                </VBox.margin>
            </ChoiceBox>
            <Button fx:id="stepButton" mnemonicParsing="false"
onAction="#madeOneStep" prefHeight="26.0"
                    prefWidth="125.0" text="Step">
                <VBox.margin>
                    <Insets bottom="60.0"/>
                </VBox.margin>
                <font>
                    <Font size="16.0"/>
                </font>
            </Button>
            <TextArea fx:id="logTextField" editable="false"
prefHeight="320.0" prefWidth="308.0" promptText="Log window"
                      stylesheets="@../css/textarea.css">
                <VBox.margin>
                    <Insets/>
                </VBox.margin>
         </TextArea>
        </VBox>
    </HBox>
</Vbox>
```

Файл pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>etu.leti</groupId>
    <artifactId>Miner</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <maven.compiler.source>14</maven.compiler.source>
        <maven.compiler.target>14</maven.compiler.target>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-controls</artifactId>
            <version>14</version>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-fxml</artifactId>
            <version>14</version>
        </dependency>
        <dependency>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-archetype-fxml</artifactId>
            <version>0.0.5</version>
        </dependency>
        <dependency>
            <groupId>org.jetbrains</groupId>
            <artifactId>annotations</artifactId>
            <version>19.0.0</version>
        </dependency>
        <dependency>
            <groupId>com.google.code.gson</groupId>
            <artifactId>gson</artifactId>
            <version>2.8.6</version>
        </dependency>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.13</version>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.0</version>
                <configuration>
                    <release>11</release>
```

```
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.openjfx</groupId>
                <artifactId>javafx-maven-plugin</artifactId>
                <version>0.0.4</version>
                <configuration>
                    <mainClass>etu.leti.gui.Main</mainClass>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```