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

ОТЧЕТ

по учебной практике

Тема: Scout

Студент гр. 8303 _____ Гришин К.И.

Студент гр. 8303 _____ Журбин К.А.

Студент гр. 8303 _____ Курлин Н.Н.

Руководитель _____ Фирсов М.А.

Санкт-Петербург

2020

ЗАДАНИЕ
НА УЧЕБНУЮ ПРАКТИКУ

Студент Гришин К.И. группы 8303

Студент Журбин К.А. группы 8303

Студент Курлин Н.Н. группы 8303

Тема практики: scout

Задание на практику:

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

Алгоритм: Дейкстра, A*.

Сроки прохождения практики: 29.06.2020 – 12.07.2020

Дата сдачи отчета: 08.07.2020

Дата защиты отчета: 08.07.2020

Студент _____ Гришин К.И.

Студент _____ Журбин К.А.

Студент _____ Курлин Н.Н.

Руководитель _____ Фирсов М.А.

АННОТАЦИЯ

Основной целью работы является получение навыков программирования на языке Java и освоение парадигмы объектно-ориентированного программирования. Цель достигается командной работой путём разработки программы с графическим интерфейсом, использующей указанные в задании алгоритмы. В этой работе необходимо реализовать алгоритм, находящий кратчайший путь от заданной точки (скаута) до всех отмеченных пользователем сундуков (один и более). Все объекты вносятся на карту пользователем взаимодействием с оконным приложением. Для проверки корректности программы используются JUnit тесты.

SUMMARY

The main goal of the work is to get programming skills in the Java language and mastering of object-oriented programming paradigm. The goal is achieved by teamwork by developing a program with a graphical interface that uses the algorithms specified in the task. In this work, it is necessary to implement an algorithm that finds the shortest path from a given point (scout) to all chests marked by the user (one or more). All objects are brought to the map by the user by interaction with the window application. JUnit tests are used to verify the correctness of the program.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Задача практики состоит в разработке приложения, позволяющего на карте в оконном приложении задавать время прохождения каждого типа ландшафта, изменять размер и ландшафт на карте, устанавливать скаута на карту и один или более сундуков. Программа должна позволять пользователю находить кратчайший путь от скаута до всех сундуков пошагово либо сразу.

Для поиска кратчайшего пути до одной цели (если сундук один или от предпоследнего сундука до оставшегося) используется алгоритм A* с эвристической функцией манхэттенского расстояния, так как движение скаута возможно только в четырёх направлениях.

Для поиска кратчайшего пути до нескольких целей (две и более) используется алгоритм Дейкстры.

Если сундук на карте один, то используется алгоритм A*. Если сундуков более одного, то сначала алгоритмом Дейкстры находится путь до ближайшего сундука, а из него – снова алгоритмом Дейкстры до оставшихся непосещёнными. Когда остаётся только один непосещённый сундук, применяется алгоритм A*.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

1.1.1. Требования к вводу исходных данных.

- Установка скаута на карту
- Установка сундука(ов) на карту
- Возможность загрузки карты из сохранения
- Возможность изменять размеры поля и ландшафты на нём
- Возможность изменять время прохождения каждого типа местности

1.1.2 – Требования к визуализации

- Ясность и удобство для пользователя (рис. 1, 2, 3)
- Выделение просмотренных вершин при пошаговом поиске пути

1.1.3 – Требования к выходным данным

- Возможность сохранения карты
- Путь должен быть отображён на карте в виде ломаной

Рисунок 1. Меню карты

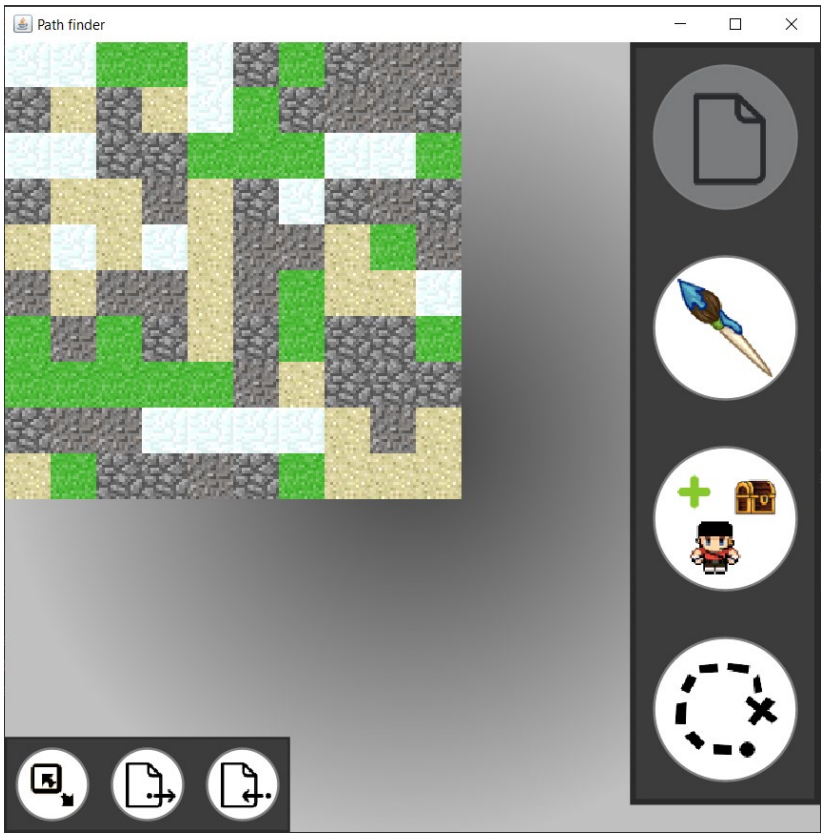


Рисунок 2. Меню инициализации

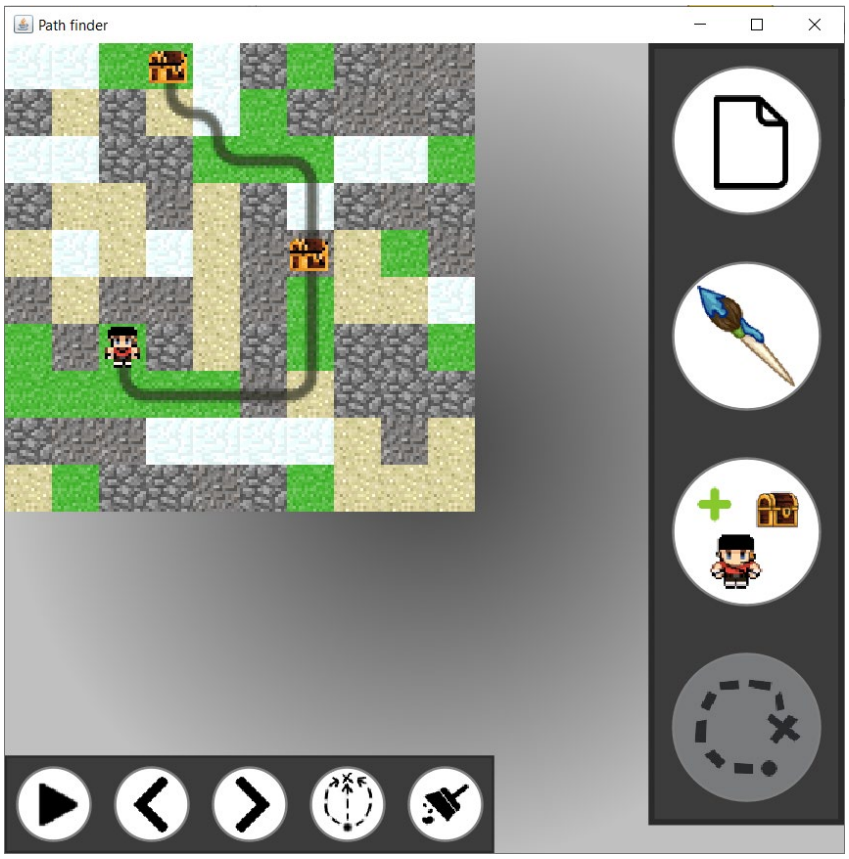


Рисунок 3. Меню запуска

1.2. Уточнение требований после сдачи прототипа и первого этапа

1.2.1. Требования к вводу исходных данных.

- Установка скаута на карту
- Установка сундука(ов) на карту
- Возможность загрузки карты из сохранения
- Возможность изменять размеры поля и ландшафты на нём
- Возможность изменять время прохождения каждого типа местности

1.2.2 – Требования к визуализации

- Ясность и удобство для пользователя (рис 1, 2, 3)
- Выделение просмотренных вершин при пошаговом поиске пути
- Возможность выполнять шаг назад алгоритма

  Небольшие изменения в иконках кнопок.

1.2.3 – Требования к выходным данным

- Возможность сохранения карты

- Путь должен быть отображён на карте в виде ломаной

1.3. Уточнение требований после сдачи второго этапа

1.3.1. Требования к вводу исходных данных.

- Возможность загрузки карты из сохранения
- Возможность изменять размеры поля и ландшафты на нём
- Возможность изменять время прохождения каждого типа местности

1.3.2 – Требования к визуализации

- Ясность и удобство для пользователя (рис 1, 2, 3)
- Выделение просмотренных и посещённых вершин при пошаговом поиске пути
- Возможность выполнять шаг назад алгоритма

  Небольшие изменения в иконках кнопок.

1.3.3 – Требования к выходным данным

- Возможность сохранения карты
- Путь должен быть отображён на карте в виде ломаной
- Закругление пути и разные цвета для разных путей одинаковой длины

2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки

1. К 30 июня обсудить стратегию поиска кратчайшего пути, создать структуру данных для хранения поля, реализовать алгоритмы A* и Дейкстры, начать работу над пользовательским интерфейсом.
2. К 1 июля оснастить графический интерфейс кнопками управления и создать структуру данных для отрисовки поля.
3. К 3 июля добавить возможности вызова диалоговых окон для изменения размера поля, параметров типа ландшафта и сохранения и загрузки. Пошаговое исполнение алгоритма. Подготовить прототип и спецификацию проекта.
4. К 4 июля добавить меню работы с алгоритмом и возможность совершать шаг назад. Пробное слияние логики и интерфейса.
5. К 5 июля создать графическую оболочку для логики, добавить структуры, конвертирующие набор точек в линии, представляющие путь.
6. К 6 июля добавить остановку алгоритма при пошаговом поиске при достижении сундука, исключить возможность запуска алгоритма по второму кругу. Добавить логирование.
7. К 7 июля закончить основную работу по логике и интерфейсу программы. Начать написание отчёта и приступить к написанию тестов.
8. К 8 июля завершить разработку всех частей проекта.

2.2. Распределение ролей в бригаде

Гришин Константин – лидер, фронтенд

Журбин Кирилл – алгоритм

Курлин Никита – тестирование, документация.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

class Application - главный класс приложения,
представляет собой "Посредника", между основными рабочими модулями.

- Создает класс отрисовки динамических объектов – Canvas
- Создает класс пользовательского интерфейса – UI
- Создает классы LandscapeListener, FileListener, ObjectsListener и AlgorithmListener, и инициализирует ими UI. Эти классы являются звеном, соединяющим интерфейс и логику.
- Создает класс карты - TileMap. Который в свою очередь является динамическим объектом. Этот класс является динамическим объектом, который изменяется от действий пользователя и требует перерисовки каждый квант времени, который установлен программой.
- Класс наследуется от JPanel, который принадлежит библиотеке Swing, следовательно, может ловить события, взаимодействия мышкой и клавиатурой.

  Каждый раз, когда мышка движется по окну, приложение получает положение мышки и указывает на клетку карты, на которую указывает курсор.

  Каждый раз, когда мышка движется с зажатой левой кнопкой мыши, приложение замеряет расстояние с прошлого события и смещает "Камеру" класса Canvas.

  Каждый раз, когда происходит клик мышью, этот клик делегируется интерфейсу, если клик попал по объектам интерфейса, то в зависимости от нажатой кнопки вызывается соответствующий Listener

  Если клик не попал в интерфейс, то высчитывается клетка поля, по которой был совершен клик и эта клетка напрямую отправляется на обработку соответствующим Listener'ам.

- Application также наследует интерфейс DialogRaiser, это значит, что приложение может принимать диалоговые окна.
  Каждый раз, когда приложение просят принять такое окно, то все события делегируются этому окну, пока то не закроет пользователь.
  Как только окно получило статус закрытого, события снова продолжает принимать класс Application.

class UI - класс пользовательского интерфейса
представляет собой "Наблюдателя" и сообщает о смене своего состояния соответствующим Listener’am

- Класс создает четыре кнопочные панели:
  Main - основная кнопочная панель и отличается от всех прочих тем, что не имеет своего Listenera
  Дополнительные: FilePanel, LandscapePanel, ObjectsPanel, AlgorithmPanel - панели являются дополнительными, такие панели имеют свой фиксированный ID и пользователь должен установить Listener для каждой такой панели, используя ее ID.
- Использование класса заключается в сообщении ему неких координат.
  Класс определяет, принадлежат ли эти координаты основной панели или текущей активной дополнительной панели.
  Если такие координаты принадлежат основной, то в зависимости от нажатой кнопки, ставится соответствующая активная дополнительная панель.
  Если координаты принадлежат текущей активной дополнительной панели, то при нажатии какой либо кнопки вызывается Listener, который принадлежит данной панели с указанием функции нажатой кнопки

class Listener - интерфейс, которым оперирует UI, чтобы делегировать запросы, касающиеся различных областей программы.

- Реализации этого класса как правило принимают в конструктор поле - TileMap, с которым они работают. Но также могут запрашивать и DialogRaiser - интерфейс, который принимает обязанности обработки диалоговых окон.
- Интерфейс содержит в себе методы notify и drop, которые вызываются UI
  notify сообщает о том, что была нажата какая-либо кнопка
  drop - о том, что можно сбросить параметры Listener'a, если это нужно
  notify определяет, может ли Listener обработать такую функцию и в случае чего либо меняет TileMap, либо вызывает диалоговое окно, которое запрашивает дополнительную информацию у пользователя.

class TileMap - графическая модель поля. Является наследником класса Field, который содержит в себе всю логику программы.

- Принимает команды по инициализации поля или запуску алгоритма. Причем вторые исполняются только тогда, когда поле инициализировано.
- Каждый раз когда алгоритм оказывает влияние на состояние поля, это отображается графически

Итоговый путь события вызванного пользователем:

- Поток действия пользователя, совершается, когда пользователь нажимает на кнопки интерфейса

Application -----> UI ----------> Listener ----------> Dialog ----------> TileMap
click call listener call dialog affects on map

- Поток отображения поля, вызывается каждый определенный квант времени

Application --------> Canvas
draw map

3.1. Структуры данных

1. Field - класс игрового поля, содержащий алгоритмы для поиска кратчайших путей и следующие поля:
   - protected transient Tile[][] fieldTiles - карта клеток
   - protected int width – ширина поля
   - protected int height – высота
   - private transient boolean algIsWork - флаг работы алгоритма поиска пути
   - private transient boolean isAlgManyTargetIsWork - флаг работы алгоритма поиска пути до нескольких сундуков
   - private transient boolean isAStar - флаг выполнения алгоритма A*
   - private transient ArrayList<Cell> notVisitedCells - список непосещенных клеток
   - private transient ArrayList<Cell> path - путь (кратчайший) до одного сундука
   - private transient ArrayList<Cell> fullPath - путь (кратчайший) для обходов всех сундуков
   - private transient Cell startCell - начальная координата
   - private transient Cell finishCell - конечная вершина
   - private transient ArrayList<Cell> finishCells - координаты сундуков
   - private transient HashMap<Cell, Cell> pathMap - словарь пар, входящих в путь
   - private transient Cell currentCell - текущая клетка
   - private transient int[][] minimalPathMap - карта минимальных расстояний до всех клеток
   - private transient SavesStep savesStep – переменная для сохранения предыдущих шагов
2. Cell - класс клетки, необходимой для работы алгоритма. Содержит поля:
   - private final int x

- private final int y
- private int distanceFunction - значение суммы эвристической функции для клетки и стоимости пути
- private int distance – расстояние от скатуа до текущей клетки

3. Tile - класс плитки содержащей ландшафт. Содержит поля:
    - private final int x
    - private final int y
    - private final TileType tileType - тип ландшафта клетки
    - public boolean isVisited - посещена ли клетка
    - public boolean isOpen - открыта ли клетка
4. TileMap – обёртка класса Field, которая преобразует данные в рисуемые объекты. Содержит поля:
    - transient private int tileSize = 16 - размер плитки в пикселях
    - transient private Scout scout – скаут на поле
    - transient private ArrayList<Chest> chests – список сундуков на поле
    - transient DrawablePath drawablePath – рисуемый кратчайший путь
    - transient boolean initialized = false – флаг инициализации алгоритма (примет true, когда будет установлен скаут и хотя бы один сундук)
5. DrawablePath – класс изображения кратчайшего пути, накладываемого на карту. Содержит поля:
    - int mapWidth – ширина изображения в клетках
    - int mapHeight – высота изображения в клетках
    - int tileSize – размер одной клетки в пикселях
    - BufferedImage mappedPath – изображение, на котором нарисован путь
    - GeneralPath pathLine – математичесий объект
    - Random randomizer – рандомайзер для выбора цвета пути или путей
    - Color color – цвет пути
    - boolean restrictRecolor = false – флаг ограничения смены цвета пути

6. SavesStep – класс для сохранения состояния алгоритма на предыдущих шагах. Содержит поля:
    - private final ArrayList<Tile[][]> savesFieldTiles – список сохраненных игровых полей предыдущих шагов
    - private final ArrayList<Cell> currentCell – список текущих клеток
    - private final ArrayList<ArrayList<Cell>> notVisitedCells - список списков непосещенных клеток
    - private final ArrayList<int[][]> minimalPathMap – список карт минимальных расстояний до всех клеток
    - private final ArrayList<ArrayList<Cell>> finishCells – список списков сундуков
    - private final ArrayList<ArrayList<Cell>> fullPath – кратчайший путь для обхода всех сундуков
    - private final ArrayList<HashMap<Cell, Cell>> pathMap – список пар путей.

3.2. Основные методы

1. Класс Field:
    - public void setStartCell(Cell startCell) - устанавливает начальную клетку (положение скаута)
    - public void setFinishCell(Cell finishCell) - устанавливает конечную клекту и отмечает флаг использования алгоритма A* как true
    - public void setFinishCells(ArrayList<Cell> finishCells) - устанавливает множество конечных клеток и флаг A* как false
    - public boolean nextStep() – обрабатывает следующий шаг алгоритма. Если конечная клектка очищена и флаг применения A* истинный, возвращается false – сигнал конца работы алгоритма. Если флаг A* истинный, то метод возвращает значение метода nextStepFindPath(), если ложный – значение метода nextStepFindPathManyTarget()

- public boolean nextStepFindPath() – обрабатывает следующий шаг алгоритма на основе A*. Если конечная клетка пуста, возвращается false. Если алгоритм только запущен, то очищаются статусы клеток, создаются необходимые списки и словари. Далее вызывается метод самого шага алгоритма stepFindPath(). Если алгоритм при этом свою работу закончил, то собирается кратчайший путь, очищается конечная клетка. Метод возвращает флаг того, завершён ли алгоритм
- public boolean stepFindPath() - реализует шаг поиска кратчайшего пути от одной клетки к другой. Если координаты текущей клетки совпадают с конечной, то возвращается false – флаг конца работы алгоритма. Текущей клеткой становится клетка из непосещённых клеток с наименьшим значением поля distanceFunction. Для всех соседей текущей клетки либо находится более короткий путь либо новый путь либо более длинный путь. В случае нового и более короткого пути соответствующие поля у клеток обновляются. Метод возвращает true
- private boolean nextStepFindDijkstraPath() – обрабатывает следующий шаг алгоритма Дейкстры. Если алгоритм только запущен, то очищаются статусы клеток, создаются необходимые списки и словари. Далее вызывается метод самого шага алгоритма stepFindDijkstraPath(). Если алгоритм при этом свою работу закончил, то находится ближайшая плитка, строится путь до нее. Метод возвращает флаг того, завершён ли алгоритм
- private boolean stepFindDijkstraPath() – реализует шаг алгоритма Дейкстры. Если все клетки посещены, то алгоритм заканчивает работу. Берется ближайшая клетка из списка непосещенных и для её соседей определяется, короче ли текущий путь в сравнении с уже найденным. Если путь короче или клетка ещё не была открыта, её

поля обновляются, а в карту путей добавляется новый путь. Метод возвращает true

- public void previousStep() – переходит к предыдущему шагу алгоритма. Если конечная клетка пуста и использовался A* или конечные клетки пусты и использовался Дейкстра, то метод завершает работу. Восстанавливаются клетки поля, текущая клетка, список непосещённых клеток, карта путей.

2. Класс TileMap:

- public void addScout(int x, int y) – создаёт на указанных координатах скаута и проверяет, чтобы он был один и не стоял на сундуке.
- public void addChest(int x, int y) - создаёт на указанных координатах сундук и проверяет, чтобы он не стоял на скауте
- public void run() – если алгоритм проинициализирован, то вызывает метод run класса Field и передаёт объекту класса DrawablePath найденный кратчайший путь.
- public void stepForward() – осуществляет шаг вперёд алгоритма, если он проинициализирован. Вызывает метод класса Field nextStep
- public void stepBack() – осуществляет шаг назад алгоритма. Вызывает метод класса Field previousStep.

4. ТЕСТИРОВАНИЕ

4.1. Написание UNIT Test

Для проверки корректности работы алгоритма был использован фреймворк автоматического тестирования JUnit. Тестами были покрыты основные методы класса Field: метод обработки следующего шага nextStep, метод получения пути до одного сундука getPath, метод получения пути до всех сундуков getFullPath, метод обработки следующего шага пути при нескольких целевых плитках nextStepFindPathManyTarget и метод нахождения всех кратчайших путей findAllPath. Во всех тестах использовался метод run запуска алгоритма поиска кратчайшего пути, поэтому можно сказать, что этот метод тоже косвенно протестирован. Методы isContained, setStartCell, setFinishCell, setFinishCells, nextStepFindPath, stepFindPath, nextStepFindDijkstraPath, stepFindDijkstraPath, findAllPathRecursion вызываются из перечисленных выше методов и также участвуют в тестировании.

4.2. Ручное тестирование программы

Ручное тестирование применялось для проверки корректности работы всех частей программы, не покрытых тестами. Тестировался функционал карты, её загрузка и сохранение, изменение размера и ландшафта. Также протестировано изменение времени прохождения ландшафта, корректность координат мыши, установки сундуков и скаута. Протестированы все функции алгоритма (рисунок 4).

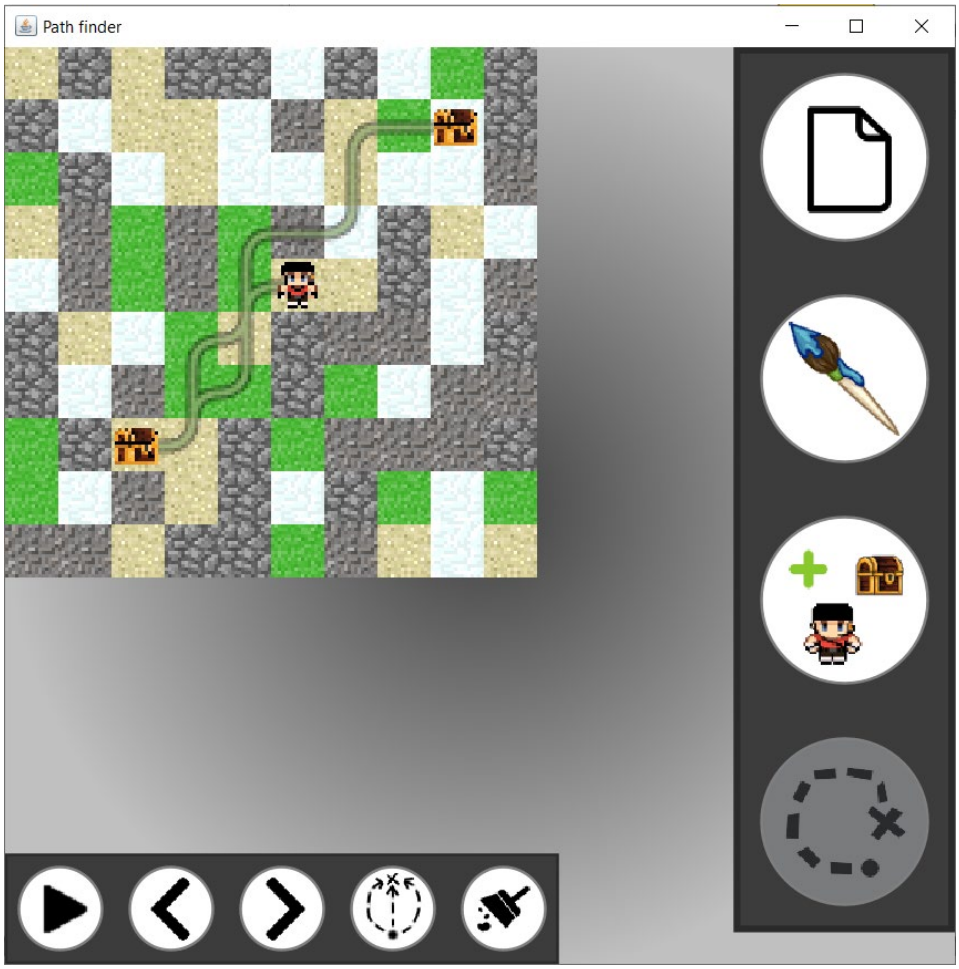


Рисунок 4. Поиск пути

На рисунке 5 продемонстрирован пример работы алгоритма, когда кажущийся кратчайшим прямой путь таковым не является. Клетка земли имеет сложность прохождения равную единице, сложность прохождения песка равна двум. Несмотря на меньшее значение эвристической функции у клеток, идущих вверх от скаута, алгоритм, опираясь на время прохождения каждого типа ландшафта, находит верный путь.

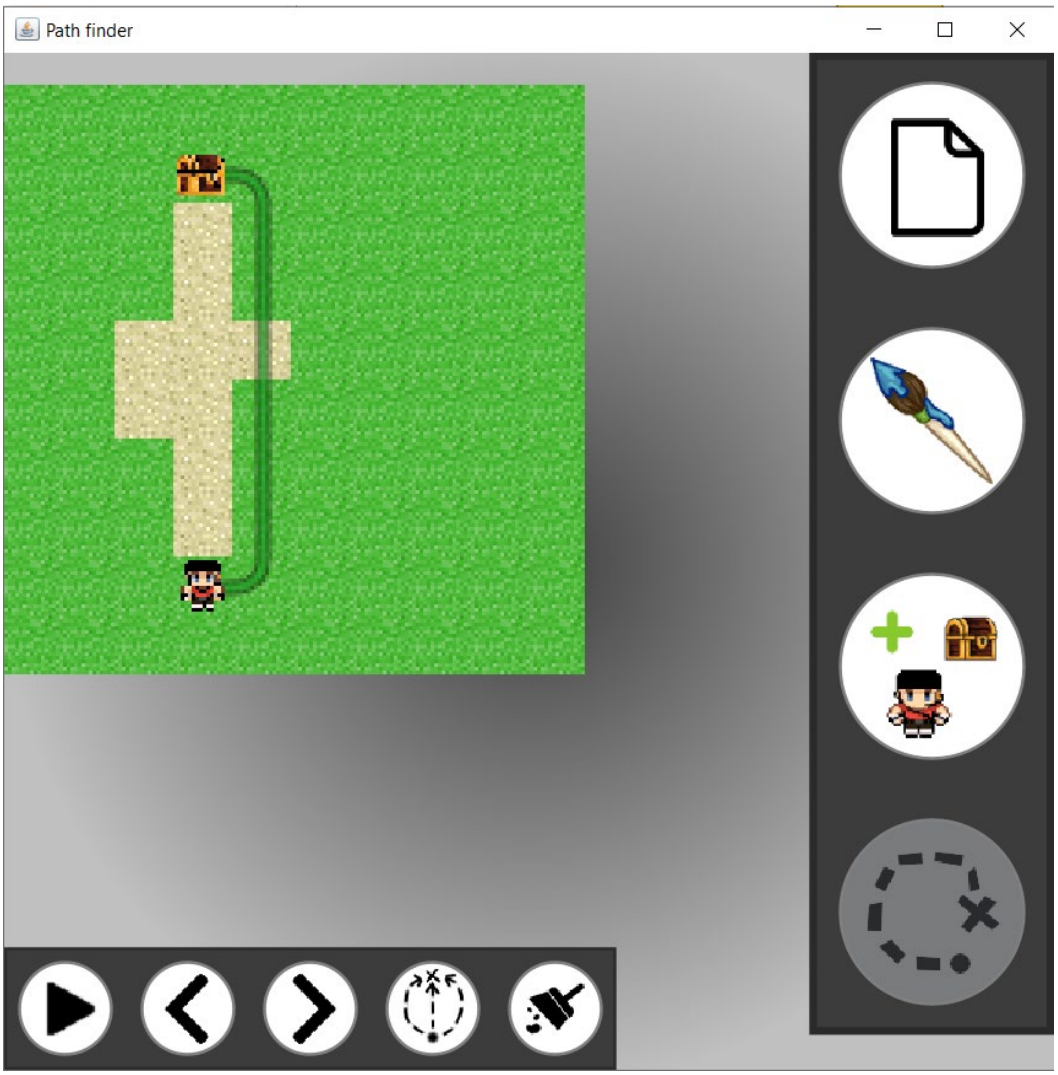


Рисунок 5. Неочевидный путь

ЗАКЛЮЧЕНИЕ

Проект завершён своевременно и успешно. Выполнены все основные требования из технического задания, а также большинство дополнительных. Пользователю представлен удобный интерфейс для демонстрации работы программы, а также для получения команд от пользователя. Приложение позволяет кликом мыши задавать позиции скаута и сундука(ов), удалять их правым щелчком мыши, загружать карты из сохранений и сохранять открытые, изменять ландшафт на карте и время его прохождения. Также добавлены возможности находить кратчайшие пути от скаута до одного или нескольких отмеченных сундуков как пошагово, так и единовременно. При поиске пути до одного сундука могут быть выведены все пути, являющиеся кратчайшими. Пути при этом имеют разные цвета и скругленную форму. При пошаговом поиске, клетки, принадлежащие открытому множеству, подсвечиваются жёлтым цветом, а закрытому – чёрным.

Работоспособность приложения основана на двух алгоритмах:

1) A* - для поиска кратчайшего пути до одной цели с эвристической функцией манхэттенского расстояния

2) Алгоритм Дейкстры - для поиска кратчайшего пути до нескольких целей (две и более).

Корректность работы приложения обусловлена тестированием.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Java Platform Standard Edition 8 Documentation // Java Documentation. URL: https://docs.oracle.com/javase/8/docs/ (дата обращения: 03.07.2020).

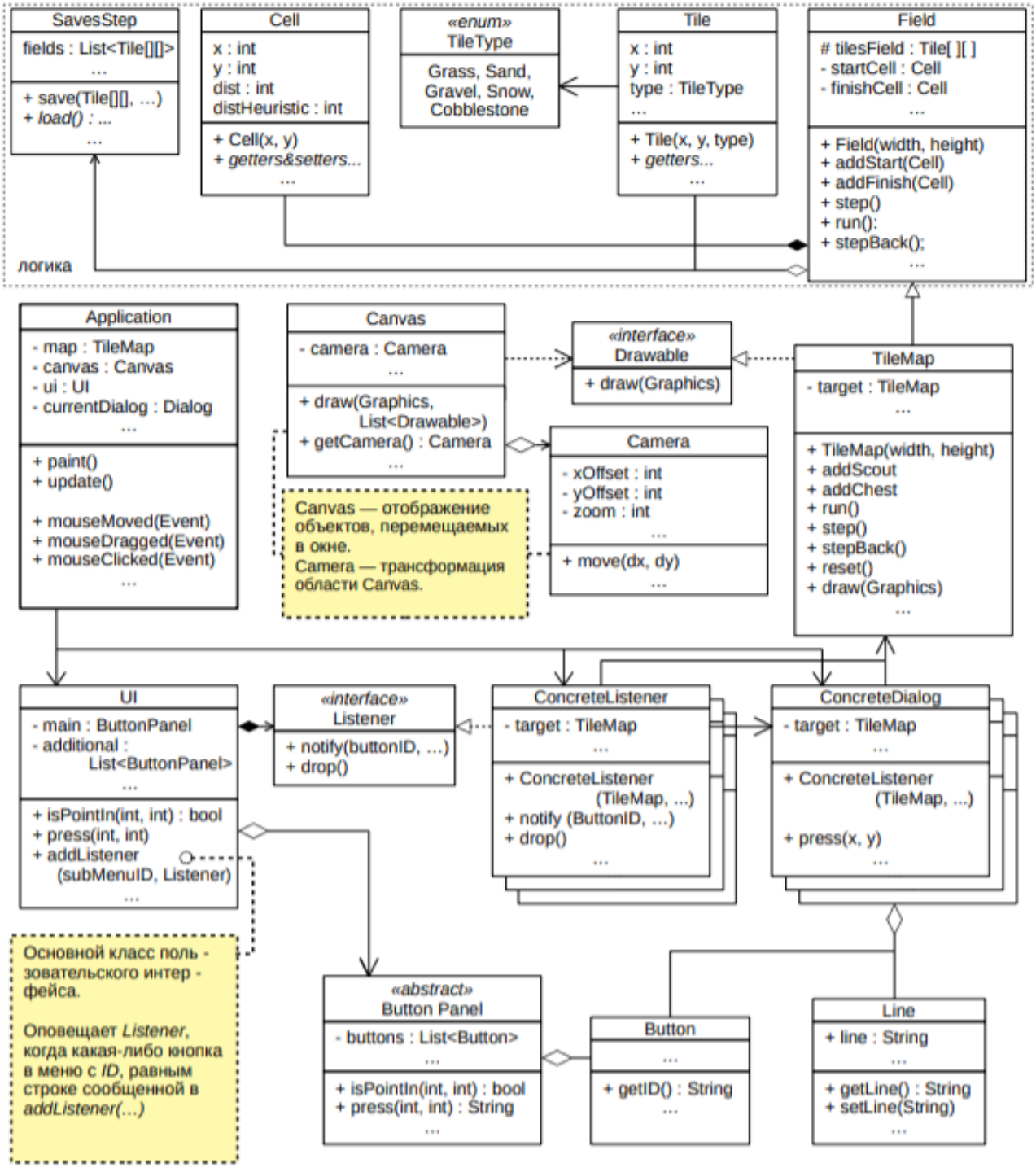
2. Шилдт Г. Java. Полное руководство. М.: Диалектика, 2018. 1488 с.

3. Учебный курс по Java на Stepik // Stepik. URL: https://stepik.org/course/187?auth=registration (дата обращения: 27.06.2020).

4. Герасимова Т.В. Учебное пособие по программированию на языке JAVA: учеб. пособие. СПб, 2006. 80 с.

ПРИЛОЖЕНИЕ А

UML-ДИАГРАММА



ПРИЛОЖЕНИЕ В
APPLICATION.JAVA

Сборка и запуск:

dir src /s /B *.java > sources.txt

javac -encoding UTF-8 -d out @sources.txt

xcopy /e assets out\assets

xcopy /e saves out\saves

cd out

java Application

Во время испольнения команды xcopy потребуется ввести ключ D

Application.java

```
import ConcreteListeners.AlgorithmListener;
import ConcreteListeners.FileListener;
import ConcreteListeners.LandscapeListener;
import ConcreteListeners.ObjectsListener;

import UI.UI;
import UI.dialog.Dialog;
import UI.dialog.DialogRaiser;

import DrawableModel.TileHighlighter;
import DrawableModel.TileMap;

import render.Canvas;
import render.Drawable;

import resources.ResourceManager;

import logic.TileType;

import javax.swing.*;
```

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;

import java.io.IOException;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;


/*
 * Application class
 * Catches system events and distributes then between modules.
 * Starts timer for update and rerender
 *
 * Logs state into default logger
 */

public class Application extends JPanel implements MouseMotionListener,
MouseListener, DialogRaiser {

    static Logger LOGGER = Logger.getLogger(Application.class.getName());


    private int renderDelay = 30;
    private int logicDelay = 100;

    private int oldMouseX = 0;
    private int oldMouseY = 0;
    private double scale = (double)
java.awt.Toolkit.getDefaultToolkit().getScreenResolution() / 96;

    Timer renderTimer;
    Timer logicTimer;
```

```
    double time = System.currentTimeMillis();

    BufferedImage background;       // gradient image behind canvas
    render.Canvas canvas;
    TileHighlighter highlighter;    // frame around tile

    UI ui;
    Dialog dialog = null;

    FileListener fileListener;
    LandscapeListener landscapeListener;
    ObjectsListener objectsListener;
    AlgorithmListener algorithmListener;

    TileMap map;
    final int tileSize = 16;

    public Application() {
        LOGGER.info("Application initiating");

        setFocusable(true);
        setFocusTraversalKeysEnabled(false);

        addMouseMotionListener(this);
        addMouseListener(this);
        addComponentListener(new ComponentListener() {
            @Override
            public void componentResized(ComponentEvent e) {
                LOGGER.info("Application size set to " + getWidth() + "x"
+ getHeight());

                ui.setSize(getWidth(), getHeight());
                LOGGER.info("UI resized");

                if(dialog != null) {
                    dialog.setPosition((getWidth() -
dialog.getWidth())/2,
                            (getHeight() - dialog.getHeight())/2);
```

```
                    LOGGER.info("Dialog \"" + dialog.getName() + "\"
moved to center");
                }

                background = new BufferedImage(getWidth(), getHeight(),
BufferedImage.TYPE_INT_RGB);
                Graphics2D g2d = (Graphics2D) background.getGraphics();
                RadialGradientPaint gradient= new
RadialGradientPaint(background.getWidth()/2, background.getHeight()/2,
                        Math.max(background.getWidth()/2,
background.getHeight()/2),
                        new float[] {0.0f, 1f}, new
Color[]{Color.DARK_GRAY, Color.LIGHT_GRAY});
                g2d.setPaint(gradient);
                g2d.fillRect(0, 0, background.getWidth(),
background.getHeight());
                g2d.dispose();
                LOGGER.info("background image is repainted");
            }

            @Override
            public void componentMoved(ComponentEvent e) { }
            @Override
            public void componentShown(ComponentEvent e) { }
            @Override
            public void componentHidden(ComponentEvent e) { }
        });

        renderTimer = new Timer(renderDelay, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                repaint();
                renderTimer.start();
            }
        });

        logicTimer = new Timer(logicDelay, new ActionListener() {
            @Override
```

```
            public void actionPerformed(ActionEvent e) {

                double delta = System.currentTimeMillis() - time;
                time += delta;

                update((int) delta);
                logicTimer.start();
            }
        });

        LOGGER.info("Create main components of application");
        canvas = new Canvas();
        LOGGER.info("Canvas is created");

        background = new BufferedImage(1, 1, BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = (Graphics2D) background.getGraphics();
        RadialGradientPaint gradient= new
RadialGradientPaint(background.getWidth()/2, background.getHeight()/2,1,
                new float[] {0.0f, 0.5f}, new Color[]{Color.DARK_GRAY,
Color.WHITE});
        g2d.setPaint(gradient);
        g2d.fillRect(0, 0, background.getWidth(),
background.getHeight());
        g2d.dispose();
        LOGGER.info("Background image is drawn");

        map = new TileMap(10, 10, 16);
        LOGGER.info("Map is created");

        try {
            ui = new UI(1024, 1024);
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(1);
        }
        LOGGER.info("UI is built");

        LOGGER.info("Create UI listeners");
```

```
        fileListener = new FileListener(this, map);
        LOGGER.info("File menu listener is created");

        landscapeListener = new LandscapeListener(this);
        LOGGER.info("Landscape menu listener is created");

        objectsListener = new ObjectsListener();
        LOGGER.info("Objects menu listener is created");

        algorithmListener = new AlgorithmListener(map);
        LOGGER.info("Algorithm neu listener created");

        ui.addListener("file_menu", fileListener);
        ui.addListener("landscape_menu", landscapeListener);
        ui.addListener("objects_menu", objectsListener);
        ui.addListener("algorithm_menu", algorithmListener);

        renderTimer.start();
        logicTimer.start();

        LOGGER.info("Application initiated");
    }


    public void paint(Graphics g){
        super.paint(g);
        g.drawImage(background, 0, 0, null);

        // compile a list of objects
        List<Drawable> objectsToDraw = new ArrayList<>();
        objectsToDraw.add(map);
        if(highlighter != null) objectsToDraw.add(highlighter);

        // draw on canvas
        canvas.draw(g, objectsToDraw);

        // draw ui
        ui.draw(g);
```

```
        if(dialog != null) dialog.draw(g);

        g.dispose();
    }


    public void update(int ms) {
        ui.update(ms);
        if(dialog != null) {
            if(dialog.closed()) dropDialog();
        }
    }


    @Override
    public void mouseDragged(MouseEvent e) {
        LOGGER.fine("mouse dragged to position (" + e.getX() * scale + ";
" + e.getY() * scale + ")");

        if(SwingUtilities.isLeftMouseButton(e)) {
            int newMouseX = (int) (e.getX() * scale);
            int newMouseY = (int) (e.getY() * scale);

            int offsetX = newMouseX - oldMouseX;
            int offsetY = newMouseY - oldMouseY;

            canvas.getCamera().move(offsetX, offsetY);

            oldMouseX = newMouseX;
            oldMouseY = newMouseY;
        }
    }

    @Override
    public void mouseMoved(MouseEvent e) {
        LOGGER.fine("mouse moved to position (" + e.getX() * scale + "; "
+ e.getY() * scale + ")");
```

```
        // get true coordinates of pixel field
        double x = e.getX() * scale / canvas.getCamera().getZoomFactor()
- canvas.getCamera().getCameraOffsetX();
        double y = e.getY() * scale / canvas.getCamera().getZoomFactor()
- canvas.getCamera().getCameraOffsetY();

        if(x >= 0 && x < (map.getWidth() * tileSize)
        && y >= 0 && y < (map.getHeight() * tileSize)) {
            int mappedX = (int)(x/tileSize);
            int mappedY = (int)(y/tileSize);

            if(highlighter == null) {
                highlighter = new TileHighlighter(mappedX, mappedY,
tileSize);
            } else {
                highlighter.setPos(mappedX, mappedY);
            }
        } else {
            highlighter = null;
        }

    }

    @Override
    public void mouseClicked(MouseEvent e) {
        LOGGER.info("mouse clicked on position (" + e.getX() + "; " +
e.getY() + ")");

        if(ui.isPointIn(e.getX(), e.getY())) {
            LOGGER.info("click on UI");
            ui.press(e.getX(), e.getY(), e);
            return;
        }

        LOGGER.info("click on canvas");

        double x = e.getX() * scale / canvas.getCamera().getZoomFactor()
```

```
- canvas.getCamera().getCameraOffsetX();
        double y = e.getY() * scale / canvas.getCamera().getZoomFactor()
- canvas.getCamera().getCameraOffsetY();

        int mappedX = (int)(x/tileSize);
        int mappedY = (int)(y/tileSize);

        LOGGER.info("point on field is (" + mappedX + "; " + mappedY +
")");

        TileType brush = landscapeListener.getBrush();
        if(brush != null) {
            map.setCell(mappedX, mappedY, brush);
        }

        if(SwingUtilities.isLeftMouseButton(e)) {
            objectsListener.setEntity(map, mappedX, mappedY);
        } else if (SwingUtilities.isRightMouseButton(e)) {
            objectsListener.removeEntity(map, mappedX, mappedY);
        }
    }

    @Override
    public void mousePressed(MouseEvent e) {
        oldMouseX = (int) (e.getX() * scale);
        oldMouseY = (int) (e.getY() * scale);
    }

    @Override
    public void mouseReleased(MouseEvent e) {
        oldMouseX = (int) (e.getX() * scale);
        oldMouseY = (int) (e.getY() * scale);
    }

    @Override
    public void mouseEntered(MouseEvent e) {
    }
```

```
    @Override
    public void mouseExited(MouseEvent e) {
        highlighter = null;
    }

    @Override
    public void raiseDialog(Dialog dialog) {
        if(dialog == null) return;

        LOGGER.info("Raised \"" + dialog.getName() + "\"");

        this.dialog = dialog;

        removeMouseListener(this);
        removeMouseMotionListener(this);
        addMouseListener(dialog);

        addKeyListener(dialog);

        dialog.setPosition(getWidth()/2 - dialog.getWidth()/2,
getHeight()/2 - dialog.getHeight()/2);
    }

    @Override
    public void dropDialog() {
        removeMouseListener(dialog);
        removeKeyListener(dialog);

        addMouseListener(this);
        addMouseMotionListener(this);

        LOGGER.info("Dropped \"" + dialog.getName() + "\"");
        dialog = null;
    }


    public static void main(String [] args) throws InterruptedException {
        initResources();
```

```
        Application.LOGGER.setLevel(Level.INFO);


        // define and initiate window
        JFrame window = new JFrame();
        Application field = new Application();

        window.setSize(700, 700);
        window.setLocationRelativeTo(null);

        window.setTitle("Path finder");
        window.setResizable(true);
        window.setMinimumSize(new Dimension(700, 700));

        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.add(field);

        window.setVisible(true);
    }


    /*
     * Initializes all field used textures
     */

    private static void initResources() {
        // landscape
        ResourceManager.loadTexture("grass", "assets/grass.png");
        ResourceManager.loadTexture("sand", "assets/sand.png");
        ResourceManager.loadTexture("snow", "assets/snow.png");
        ResourceManager.loadTexture("gravel", "assets/gravel.png");
        ResourceManager.loadTexture("cobblestone",
"assets/cobblestone.png");

        // objects
        ResourceManager.loadTexture("scout", "assets/objects/scout.png");
        ResourceManager.loadTexture("chest", "assets/objects/chest.png");
    }
```

```
}
```