Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Программирование

Отчет по курсовой работе
Игра: Лабиринт

Работу выполнил:
Мальцев М.С.
Группа: 23501/4
Преподаватель:
Вылегжанина К.Д.

Санкт-Петербург
2016

Содержание

1 Игровое приложение: Лабиринт

Лабиринты – это одно их самых древних развлечений человечества. Еще в Древнем Риме добровольцы сами уходили в лабиринты, чтобы проверить свои выносливость и смекалку. К сожалению, многие не возвращались.

Лабиринт[1] — какая-либо структура (обычно в двухмерном или трёхмерном пространстве), состоящая из запутанных путей к выходу (и/или путей, ведущих в тупик).

Под лабиринтом у древних греков и римлян подразумевалось более или менее обширное пространство, состоящее из многочисленных залов, камер, дворов и переходов, расположенных по сложному и запутанному плану, с целью запутать и не дать выхода несведущему в плане лабиринта человеку.

Лабиринты всегда были слишком загадочным и заманчивым объектом, чтобы оказаться вне волшебного мира головоломок, но прежде чем стать популярной головоломкой, которая в наше время ассоциируется со словом "лабиринт"прошло не одно тысячелетие.[2]

В современном мире головоломка лабиринт приобрела большую популярность. Можно с уверенностью утверждать, что данный тип головоломок повлиял на игровую индустрию, не только потому что существуют игры типа лабиринт, но и потому что множество приложений напрямую не относящиеся к данному игровому жанру включают себя лабиринты, например, Skyrim[3]

В связи со всем выше перечисленным, было принято решение создать игровое приложение, предоставляющее пользователю возможность прохождения лабиринтов.

1.1 Концепция игрового приложения Лабиринт

Программа представляет собой логическую игру, которая позволяет управлять существом помещённым в лабиринт.

Приложение отрисовывает игровое поле, на которе помещается существо. Пользователь может управлять им с помощью нажатия на то место, в которое он хочет переместить протагониста. Игра считается законченой, если пользователь довёл протагониста до выхода. Для усложнения задачи прохождения существуют механизмы типа ключ-дверь(в некоторые ячейки игрового поля помещены двери, сквозь которые можно пройти только при наличие ключей).

1.2 Задание

Разработать приложение под операционные системы Windows 7+ и Android, позволяющее проходить лабиринты.

1.3 Минимально работоспособный продукт

Приложение, которое предоставляет возможность пройти небольшой лабиринт.

1.4 Вывод

Пояснён выбор темы курсового проекта. Описана концепция игрового положения Лабиринт. Определено задание.

2 Проектирование игрового приложения Лабиринт

2.1 Архитектура приложения

Был использован шаблон проектирования Model-View-Presenter[4]

Его использование обусловленно тем, что:

[1] https://en.wikipedia.org/wiki/Maze
[2] http://puzzlepedia.ru/historilabirint.html
[3] https://ru.wikipedia.org/wiki/The_Elder_Scrolls_V:_Skyrim
[4] https://ru.wikipedia.org/wiki/Model-View-Presenter

- требовалось обеспечить расширяемость приложения, так как существовала некоторая неопределённость по поводу того, какую функциональность должно предоставлять приложение, так как планировалось учесть новые пожелания пользователей
- требовалось обеспечить скорость разработки
- этот шаблон интересен с учебной точки зрения

Был использован шаблон проектирования Observer[5]. Целю его применения было оповещение слушателей о событиях окончания игры, открытия двери и нахождение игроком ключа.

Архитектура Модели выглядит следующим образом:

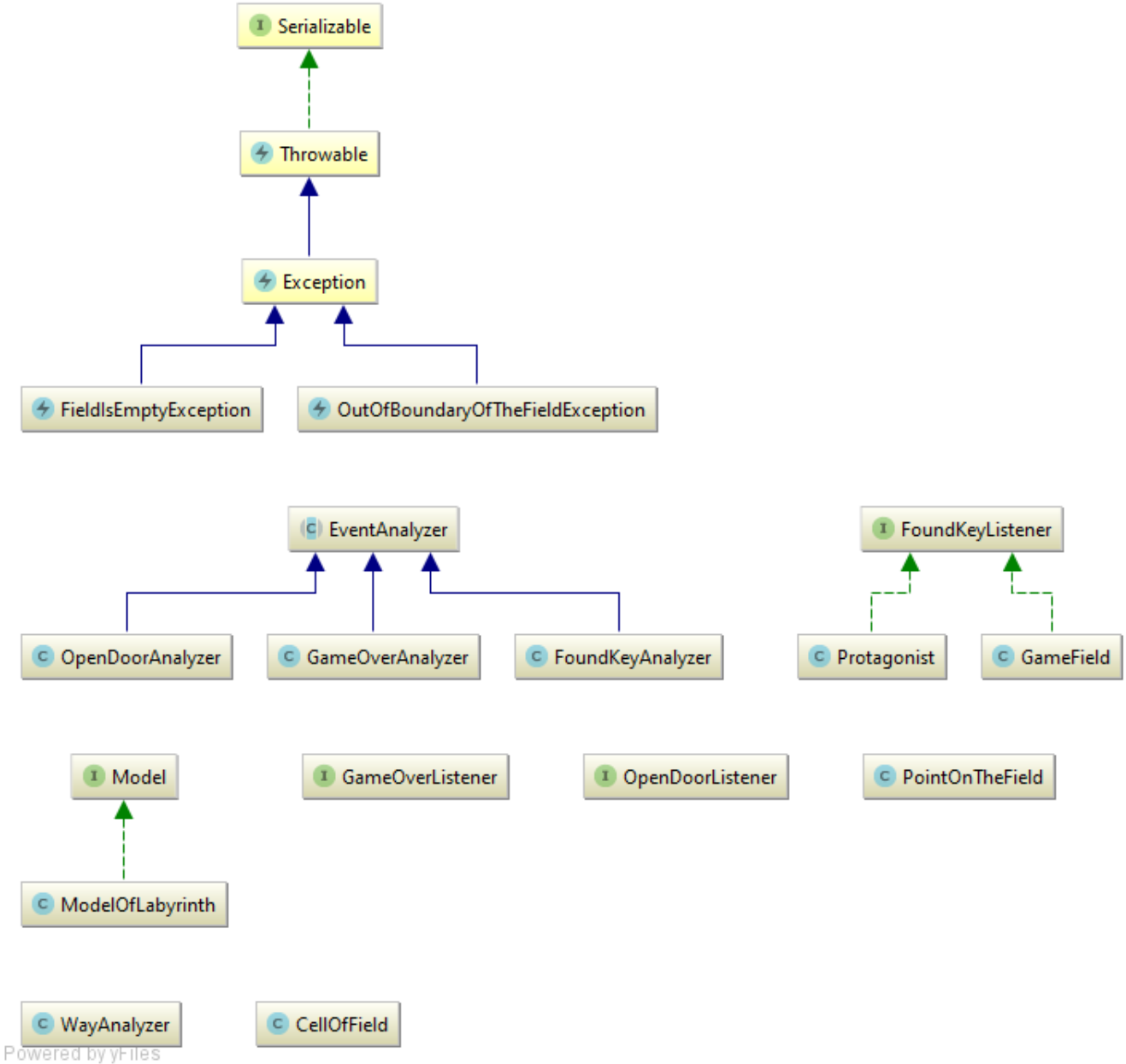

Рис. 1: Диаграма классов Модели

Модель предоставляет следующую функциональность:

- Получить список дверей на карте

[5]http://gameprogrammingpatterns.com/observer.html

- Получить список ключей на карте
- Получить позицию протагониста
- Подписаться на события: окончания игры, открытия двери и нахождение игроком ключа, и отписаться от них
- Передвинуть протагониста
- Установить игровое поле
- Получить проходимые ячейки поля
- Получить информацию о проходимости ячейки

2.2 Диаграмма компонентов

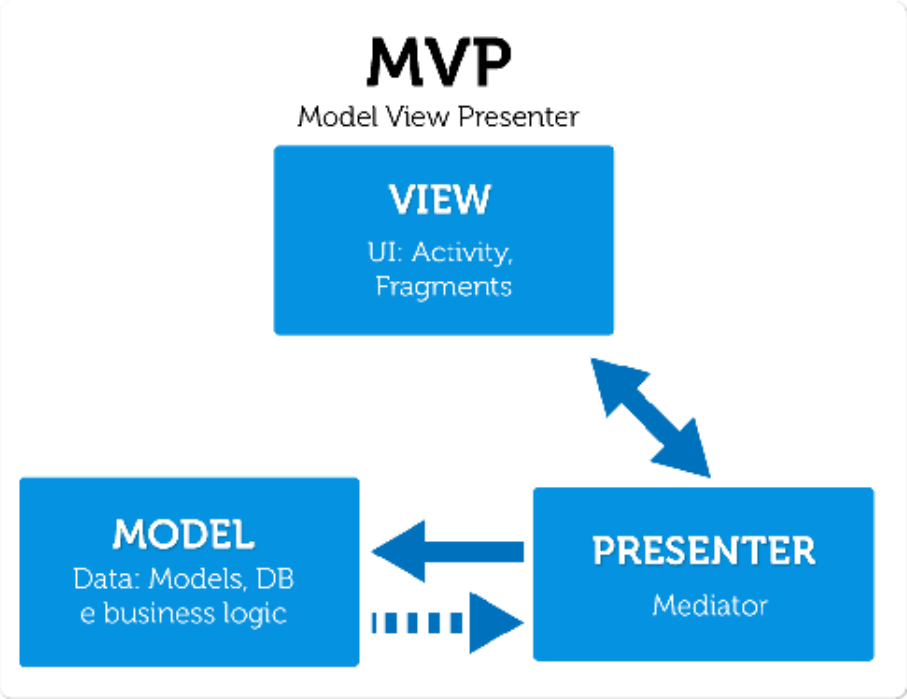

Рис. 2: Диаграмма компонентов

- Модель – содержит логическую часть игры, предоставляет данные для пользовательского интерфейса
- Представление – отвечает за взаимодействие с пользователем путём отрисовки изображения на экране и фиксирования команд и событий, которы впоследствии перенаправляет Presenter'у
- Presenter – управляет Моделью и Представлением. Указывает Представлению, что нужно отрисовывать в данный момент, принимает его оповещания о командах и сигнлах, реагирует на них и, если это необоходимо, связывается с Моделью для получения данных.

2.3 Формат задания лабиринта

Формат задания лабиринта:

s00000
10k01f
101010
101010
111d10

Где:

0 – непроходимая ячейка поля

1 – проходимая ячейка поля

s – это «start», начало пути

f – это «finish», конец пути(выход)

k – это «key», ключ

d – это «door», дверь

Проходимых клеток должно быть минимум две, иначе этот лабиринт не будет использован.

Отсчёт координат ячеек ведётся с левой верхней.

Если не указать начало или конец пути, то они выбираются автоматически из проходимых клеток, соответственно первая и последняя, по номеру индекса.

Если передать лабиринт непрямоугольной формы, то он будет автоматически приведён к прямоугольнику путём отсечения лишних символов

2.4 Вывод

Было решено использовать шаблоны проектирования Model-View-Presenter и Observer. Была описана функциональность предоставляемая Моделью. Был объяснён формат задания лабиринтов.

3 Реализация игрового приложения Лабиринт

3.1 Используемые версии

- IntelliJ IDEA 2016.3.1
  Build IU-163.9166.29
  For educational use only.
  JRE: 1.8.0 102-b14 amd64
  JVM: Java HotSpot(TM) 64-Bit Server VM by Oracle Corporation

- Java language level: 6
- Операционная система: Windows 10 x64
- LibGDX 1.9.5
- gdx-texturepacker-3.2.0
- Система автоматической сборки: Gradle 2.14

3.2 LibGDX и его использование при разработке игрового приложения

LibGDX[6] — фреймворк для создания игр и приложений, написанный на Java с использованием C и C++ (для более быстрой работы). Он позволяет писать кроссплатформенные игры и приложения используя один код. [7]

Решение, о его использование было принято по трём причинам:

1 Кроссплатформенность, именно благодаря этому параметру была достигнута цель создать приложение сразу под две операционные системы

2 Удобство создания графических объектов

3 Фреймворк интересен с учебной точки зрения

[6]https://libgdx.badlogicgames.com
[7]https://ru.wikipedia.org/wiki/LibGDX

Большая часть его функциональных возможностей, описанных по ссылке[8], не использовалась, так как задача была максимально сузить его влияние на остальные части кода, помимо View. В итоге он не используется в Модели, а Presenter пользуется лишь пакетом com.badlogic.gdx.files, чтобы сделать чтение файла кроссплатформенным

3.3 Процесс разработки игрового приложения

Было проведено первичное знакомство с LibGDX и создано приложение, в котором почти не участвует Модель, но отображается протагонист на экране. После полученя опыта работы с LibGDX и принятия решения, что этот фреймворк подходит для решаемой задачи было решено заняться непосредственно развитием функциональности Модели. В итоге выбраный путь позволил корректировать Модель во время разработки таким образом, чтобы с ней было удобно работать, а UI в свою очередь позволил удобно использовать Модель.

На следующих изображениях поэтапно приведён процесс разработки приложения:

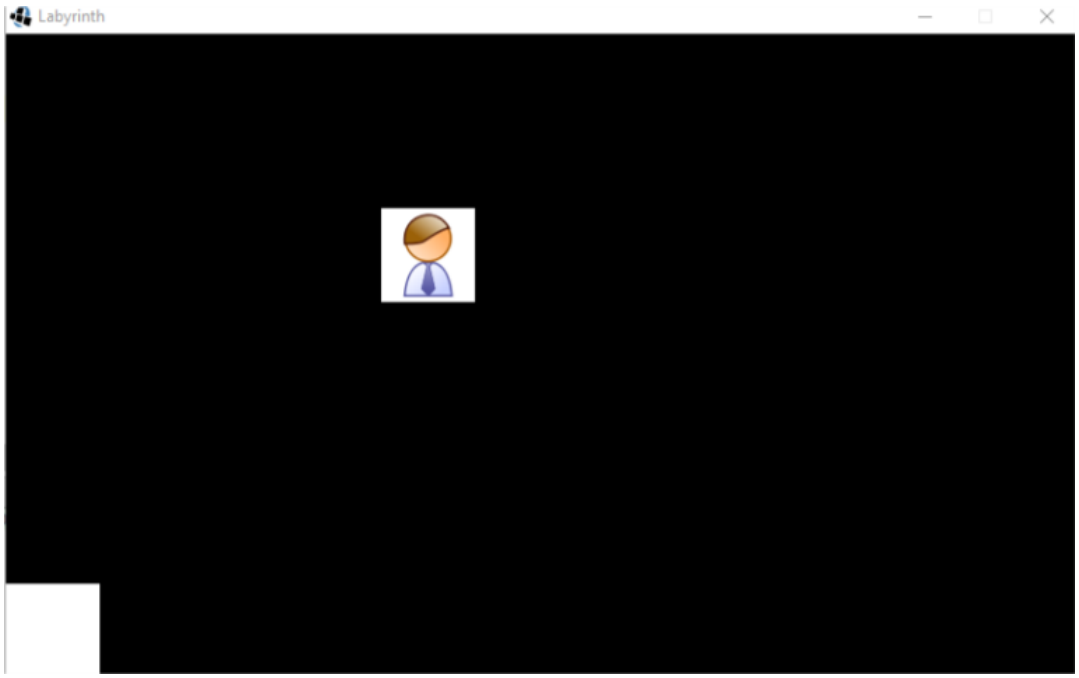

Рис. 3: Снимок экрана иллюстрирующий вид пользовательского интерфейса на начальном этапе работы приложения

На рисунке 3 изображён экран, на котором отрисовано изображение протагониста без участия Модели, поле, две белые клетки, которое задётся из Модели. Есть возможность перемещения героя между этими двумя клетками, в остальные части протагонист переместиться не может.

[8]http://www.libgdx.ru/2013/08/introduction.html

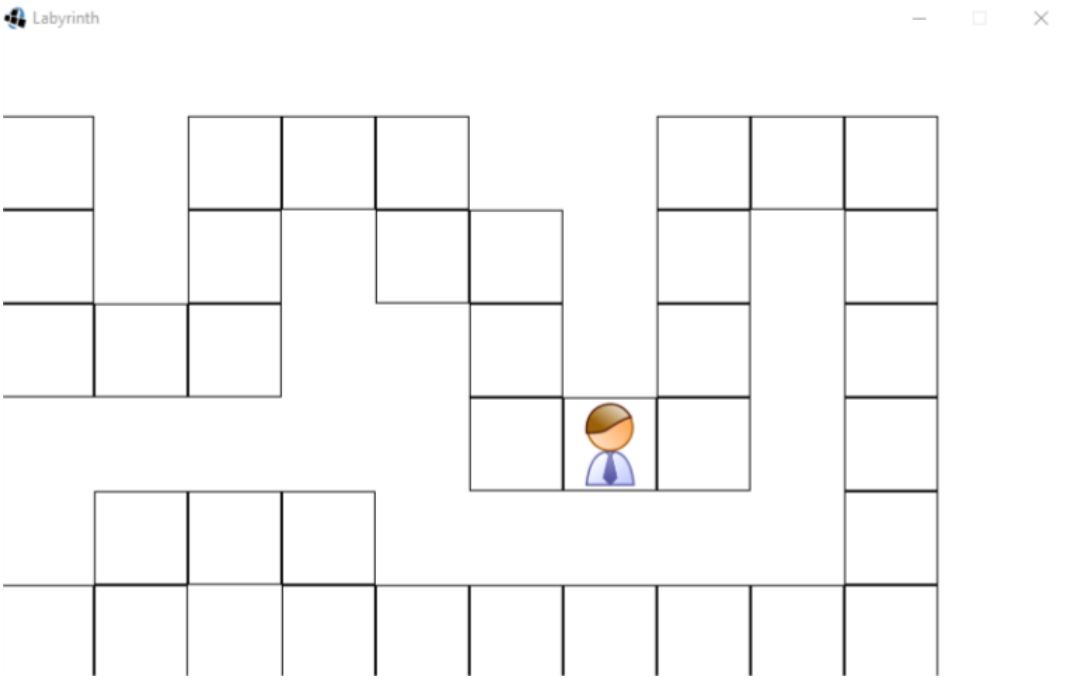

Рис. 4: Снимок экрана иллюстрирующий вид пользовательского интерфейса после создания игрового поля

На рисунке 4 в отличие от рисунка 3, поле задаётся путём передачи Модели определённого параметра. И было решено поменять цвет фона на белый. Протагонист может переместиться в любую проходимую точку поля.

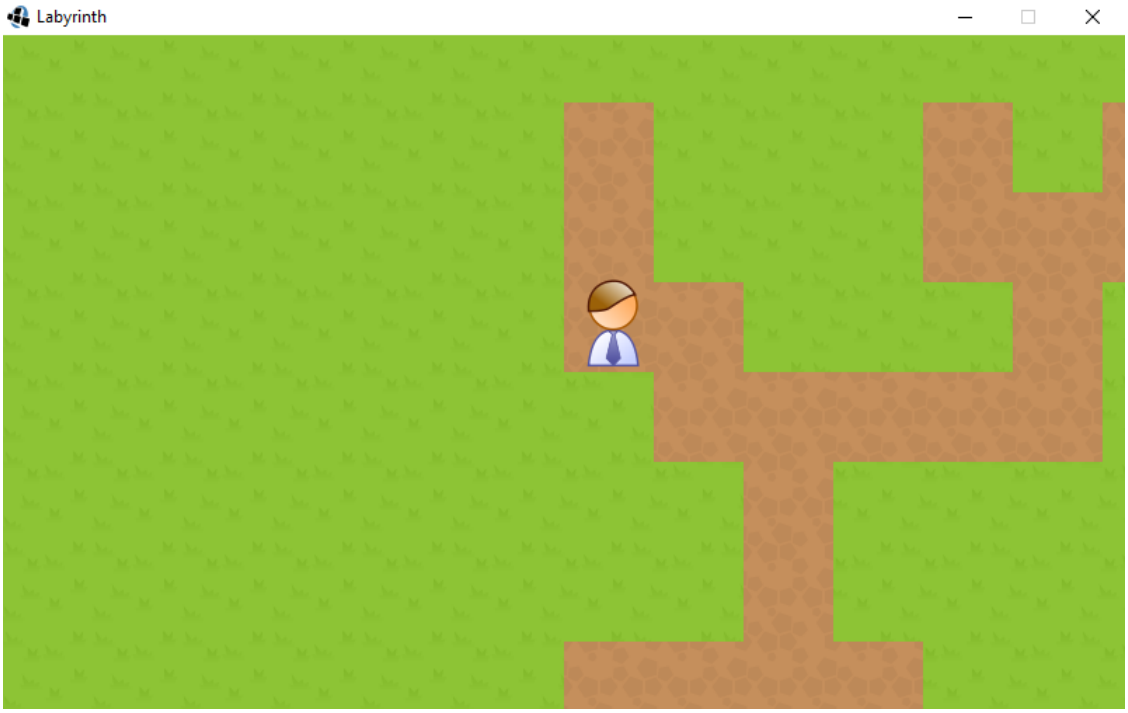

Рис. 5: Наложение текстур

Использовались текстуры с открытого источника[9].

[9]http://kenney.nl/assets

Рис. 6: Добавление главного экрана

Было решено добавить главный экран для приложения.

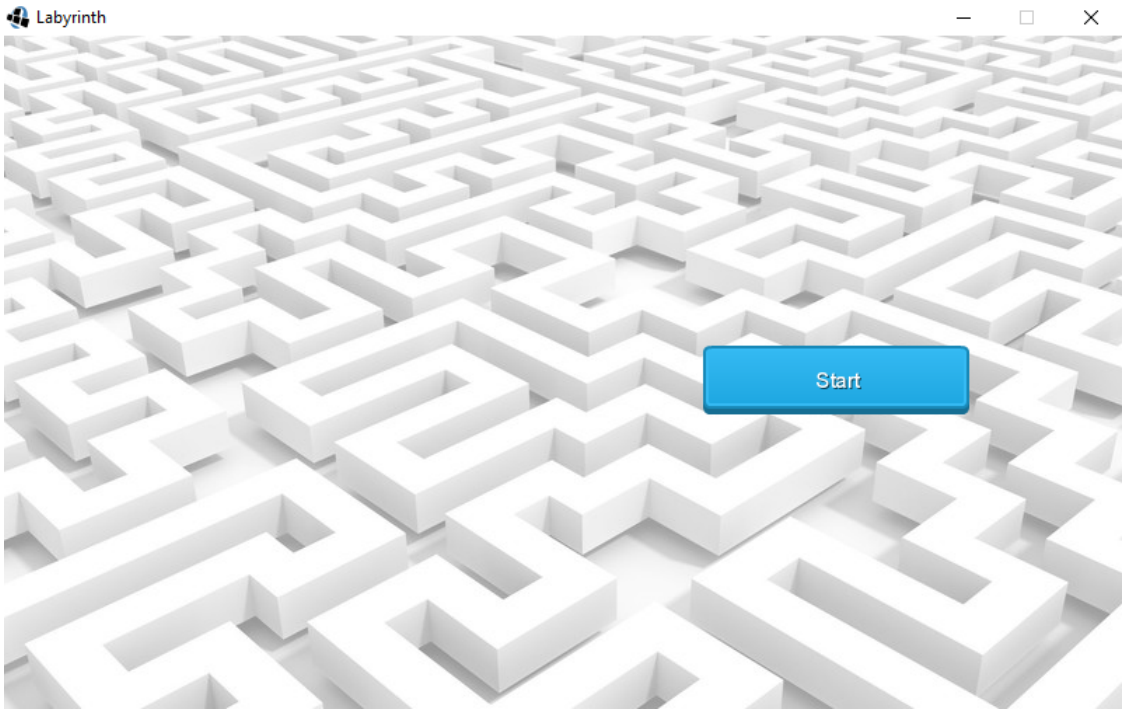

Рис. 7: Декорирование главного экрана

Оформление главного экрана было сделано с учётом тематики приложения.

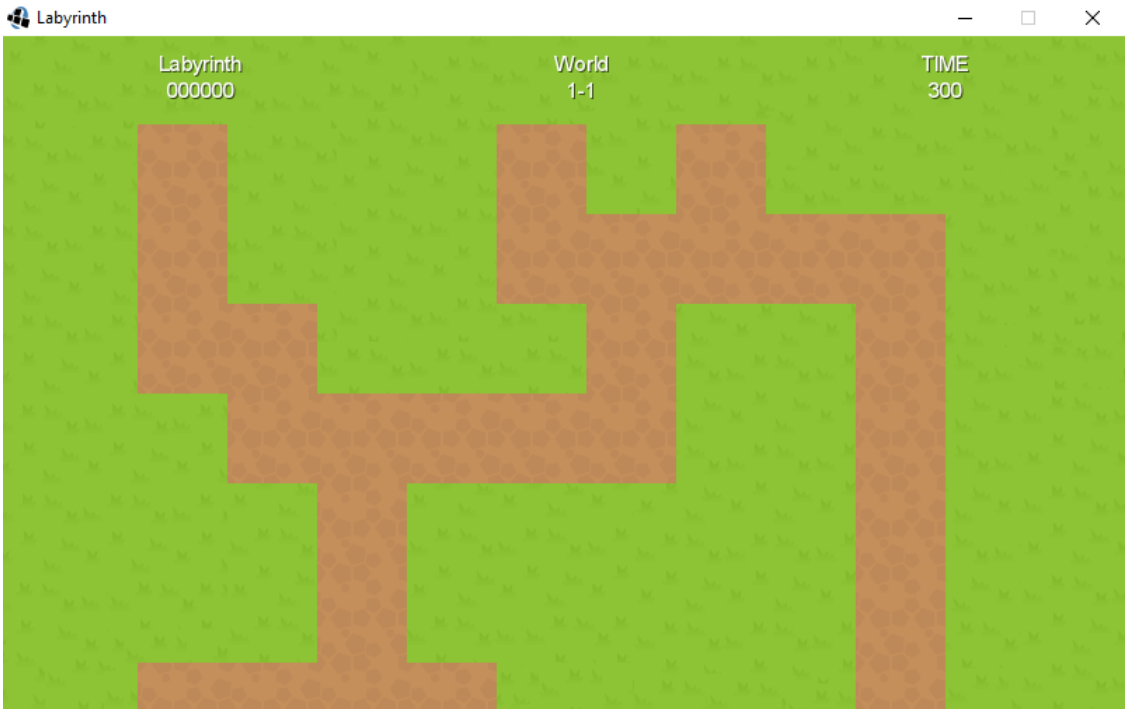

Рис. 8: Добавление информационной панели

Добавлена информационная панель, на которую выводится информация для пользователя.

Рис. 9: Обозначение выхода из лабиринта

Выход из лабиринт обозначен специальной для него текстурой, которая была взята из уже упомянутого открытого ресурса.

Рис. 10: Реакция приложения на событие прохождение лабиринта

На рисунке 10 изображён процесс реагирования приложение, на событие прохождение лабиринта. Протагонист помещён в ячейку поля, которая считается выходом из лабиринта, а приложение выводит в консоль сообщение о том, что игра окончена.

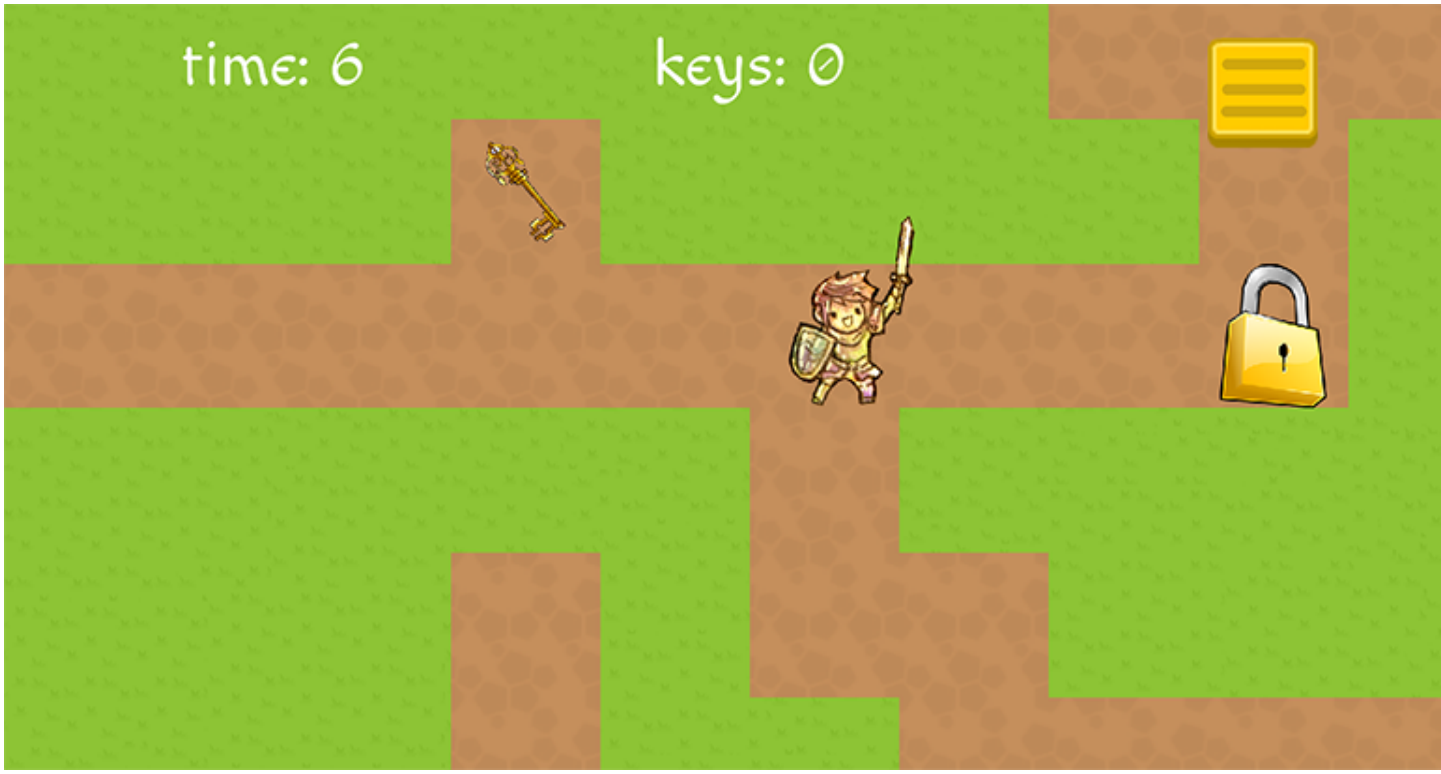

Рис. 11: Окончательный вид игрового приложения

В приложение были добавлены ключи и двери. В панель информации выводится время от начала

игры, количество ключей, а так же там размещена кнопка выхода в главное меню приложения.

3.4 Перспективы развития приложения

Планируется реализовать следующую функциональность:

- добавить миникарту, с отображением уже пройденого пути
- добавить сюжетную линию
- реализовать плавное перемещение протагониста, вместо перемещения рывками, в том числе и добавления анимации
- и т.п.

3.5 Вывод

Были описаны используемые средства разработки. Кратко описан фреймворк LibGDX и обосновано его использование. Был поэтапно описан процесс разработки приложения.

4 Процесс обеспечения качества и тестирование игрового приложения Лабиринт

4.1 Просмотр кода

Написаный код просматривался двумя людьми(Иван Крылов 16.12.2016 и ...) неучаствующими в его создании:

- 10 замечаний[10]
- 143 замечания

4.2 Демонстрации

Было проведено три демонстрации приложения

Во время демонстраций были высказаны следующие пожелания:

1 демонстрация: Начать писать логику приложения

2 демонстрация: Добавить ключи и двери

3 демонстрация: Поменять текстуры ключей и дверей, добавить выход из игрового процесса в главное меню

4.3 Автоматические тесты

При разработке приложения были написаны автоматические тесты. Это значительно ускорило разработку приложения. С их помощью были выявленны такие ошибки как:

- Неправильное построение пути между точками
- Некорректная инициализация игрового поля
- Проблемы добавления нового наблюдателя за событием и проблемы его оповещения
- и т.п.

Сценарий автоматических тестов, был следующий:

1 Создать экземпляр тестируемого класса

2 Поочереди проверить правильность работы всех механизмов класса, путём сравнивания с ожидаемым результатом

[10] https://github.com/mikle9997/Course-work/commit/fb61e9d6a3187a90e0aeacf34cb2c7dbcf346144

3 Проверить выводимый результат(информация, о том сколько тестов прошло, а сколько упало)

4 Радоваться, если всё хорошо или исправить упавшие тесты

На изображении окно интегрированой среды разработки с результатами исполнения тестов

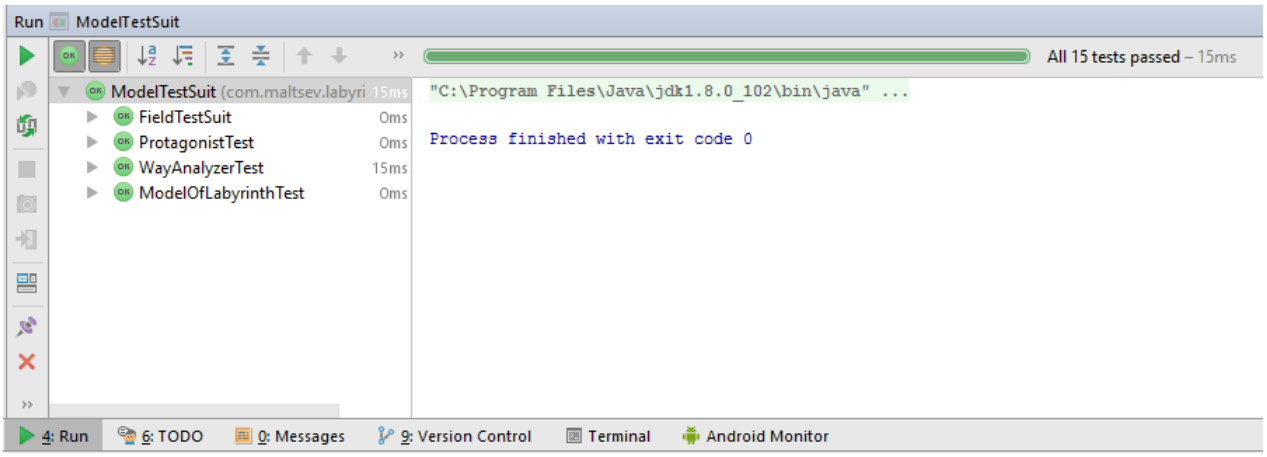

Рис. 12: Процесс тестирования

На следующих рисунках показан процент покрытия Модели тестами.

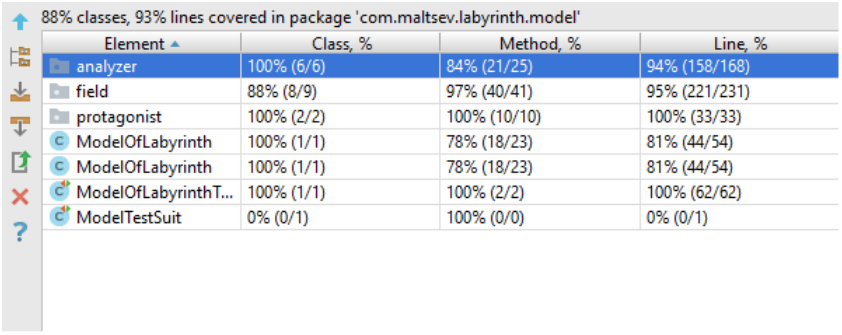

Рис. 13: Таблица процентов покрытия кода для Модели

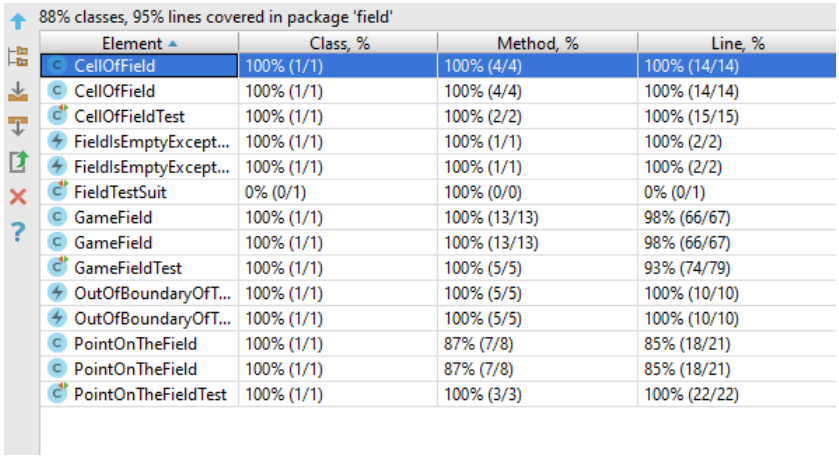

Рис. 14: Таблица процентов покрытия кода для пакета field из Модели

Как показали тесты процент покрытия классов из Модели от 78-100%

4.4 Ручное тестирование

Руками тестирование проводилось по следующему сценарию

- Запустить приложение
- Нажать на кнопку "Start"в главном меню
- Нажать на точку, в которую мне хочется переместиться и проверить переместился ли протагонист туда
- Собать ключ, путём помещения протагониста на клетку, с ключом и проверь отображение ключа в верхней панеле
- Открыть дверь, путём помещения протагониста с ключом на клетку двери
- Поместить протагониста на клетку выхода из лабиринта и проверить экран на появление панели выхода из игры
- Проверить вышло ли приложение на главный экран
- Сново нажать на конпку "Start"
- Нажать на конпку выхода в меню и проверить вышло ли оно или нет

4.5 Нереализованая функциональность

На данный момент:

- останавливать таймер после конца игры
- выводить результат таймера на панель конца игры
- добавить спрайты изображений для плавности перехода
- добавить экран паузы

4.6 Вывод

Были описаны способы обеспечения качества и тестирования, а также описана нереализованая функциональность.

5 Выводы

Было разработано игровое приложение Лабиринт. Был изучен сторонний фреймворк LibGDX и патерн проектирования Model-View-Presenter. Созданое в ходе работы приложение было протестировано, также были определены возможные перспективы развития функциональности приложения. В дальнейшем планируется продолжение поддержки и улучшение приложения, а также исправление текущих недочётов.

6 Приложение

Исходный ход можно найти в репозитории[11] на ресурсе GitHub

6.1 Листинги

```
1  package com.maltsev.labyrinth.model;
2
3
4  import com.maltsev.labyrinth.model.analyzer.event.gameover.GameOverListener;
5  import com.maltsev.labyrinth.model.analyzer.event.keysanddoors.doors.OpenDoorListener;
6  import com.maltsev.labyrinth.model.analyzer.event.keysanddoors.keys.FoundKeyListener;
7  import com.maltsev.labyrinth.model.field.FieldIsEmptyException;
8  import com.maltsev.labyrinth.model.field.PointOnTheField;
9
10 import java.util.ArrayDeque;
11 import java.util.List;
12
13 /**
14  * ModelOfLabyrinth рассматривается в качестве поставщика данных,
15  * которые будут отображаться во View.
16  */
17 public interface Model {
18
19     /**
20      * Возможно ли поместить протигониста в определённую ячейку
21      * @param x координата ячейки по оси X
22      * @param y координата ячейки по оси Y
23      * @return возвращяет значение логического типа, если true, то протагонист может находится в этой
    ↪ ячейке, иначе false
24      */
25     boolean isItPassableCells(final int x, final int y);
26
27     /**
28      * Возможно ли поместить протигониста в определённую ячейку
29      * @param point точка указывающая на ячейку
30      * @return возвращяет значение логического типа, если true, то протагонист может находится в этой
    ↪ ячейке, иначе false
31      */
32     boolean isItPassableCells(final PointOnTheField point);
33
34     /**
35      * @return Размер поля по оси X
36      */
37     int getSizeOfFieldX();
38
39     /**
40      * @return Размер поля по оси Y
41      */
42     int getSizeOfFieldY();
43
44     /**
45      * @return Массив объектов Точка на плоскости, который содержит номера ячеек, которые являются
    ↪ проходимыми
46      */
47     List<PointOnTheField> getPassableCells();
48
49     /**
50      * Установка игрового поля
51      * @param newField игровое поле, строкаматрица-, где 1,s,f - проходимые элементы, а 0 - нет, s -
    ↪ начальная точка поля, f - конечная, а новая строчка задаётся \n
52      * @throws FieldIsEmptyException выбрасывается, когда поле задано лишь 0, те.. ходить некуда
53      */
```

[11] https://github.com/mikle9997/Course-work

```
54      void setGameField(final String newField)  throws FieldIsEmptyException;
55
56      /**
57       * Перемещает протагониста в определённыю точку пространства, если это возможно, иначе оставляет на
        ↪ прежнем месте.
58       * Работает только если конечная точка пути находитсся не дальше 5 клеток от начальной
59       * @param x координата ячейки по оси X
60       * @param y координата ячейки по оси Y
61       * @return маршрутмассив( точек) перемещения из одной точки в другую, если он возможен, иначе
        ↪ null
62       */
63      @org.jetbrains.annotations.Nullable
64      ArrayDeque<PointOnTheField> movesOfProtagonist(final int x, final int y);
65
66      /**
67       * Добавляет слушателя на событие окончание игры
68       * @param listener объектслушатель–
69       */
70      void addListenerOfGameOver(GameOverListener listener);
71
72      /**
73       * Отписка слушателя от раассылки на Конец игры
74       * @param listener объектслушатель–
75       */
76      void removeListenerOfGameOver(GameOverListener listener);
77
78      /**
79       * Добавляет слушателя на событие Найден ключ
80       * @param listener объектслушатель–
81       */
82      void addListenerOfFoundKey(FoundKeyListener listener);
83
84      /**
85       * Отписка слушателя от раассылки на Найден ключ
86       * @param listener объектслушатель–
87       */
88      void removeListenerOfFoundKey(FoundKeyListener listener);
89
90      /**
91       * Добавляет слушателя на событие Открытие двери
92       * @param listener объектслушатель–
93       */
94      void addListenerOfOpenDoor(OpenDoorListener listener);
95
96      /**
97       * Отписка слушателя от раассылки на Открытие двери
98       * @param listener объектслушатель–
99       */
100     void removeListenerOfOpenDoor(OpenDoorListener listener);
101
102     /**
103      * @return Точка, местоположение героя
104      */
105     PointOnTheField getPositionOfProtagonist();
106
107     /**
108      * @return начальная точка поля
109      *
110      * Та точка, куда помещается протагонист, с самого начала игры
111      */
112     PointOnTheField getStartPosition();
113
114     /**
115      * @return конечная точка поля
116      *
117      * Точка, куда нужно пройти протагонисту, чтобы окончить игру
118      */
119     PointOnTheField getFinishPosition();
120
121     /**
122      * Установить значение дальности шага протагониста
123      *
124      * В общем, шаги были введены, чтобы игрок не мог пройти
125      * от начала карты до конца в один клик
126      * @param valueOfRangeOfStep дальность шага
127      */
```

```
128     void setValueOfRangeOfStep(int valueOfRangeOfStep);
129
130     /**
131      * @return Массив с координатами дверей
132      * Изначально двери закрыты, но если есть ключ, то дверь откроется, а
133      * количество ключей уменьшится на 1
134      */
135     List<PointOnTheField> getDoors();
136
137     /**
138      * @return Массив с координатами ключей
139      * Ключи нужны чтобы открывать двери
140      */
141     List<PointOnTheField> getKeys();
142
143     /**
144      * @return Число ключей, собраных игроком
145      */
146     int getNumberOfKeys();
147 }
```

```
1  package com.maltsev.labyrinth.model;
2
3  import com.maltsev.labyrinth.model.analyzer.WayAnalyzer;
4  import com.maltsev.labyrinth.model.analyzer.event.gameover.GameOverAnalyzer;
5  import com.maltsev.labyrinth.model.analyzer.event.gameover.GameOverListener;
6  import com.maltsev.labyrinth.model.analyzer.event.keysanddoors.doors.OpenDoorAnalyzer;
7  import com.maltsev.labyrinth.model.analyzer.event.keysanddoors.doors.OpenDoorListener;
8  import com.maltsev.labyrinth.model.analyzer.event.keysanddoors.keys.FoundKeyAnalyzer;
9  import com.maltsev.labyrinth.model.analyzer.event.keysanddoors.keys.FoundKeyListener;
10 import com.maltsev.labyrinth.model.field.FieldIsEmptyException;
11 import com.maltsev.labyrinth.model.field.GameField;
12 import com.maltsev.labyrinth.model.field.OutOfBoundaryOfTheFieldException;
13 import com.maltsev.labyrinth.model.field.PointOnTheField;
14 import com.maltsev.labyrinth.model.protagonist.Protagonist;
15
16 import java.util.ArrayDeque;
17 import java.util.List;
18
19 public class ModelOfLabyrinth implements Model {
20
21     public ModelOfLabyrinth() {}
22
23     /**
24      * Главный герой
25      * Здесь хранится информацию о том, где он находится и его
26      * определённые параметры например(, сколько у него ключей)
27      */
28     private Protagonist protagonist;
29
30     /**
31      * Игровое поле.
32      */
33     private GameField gameField;
34
35     /**
36      *  Наблюдатель за концом игры
37      *  Оповещает его слушателей о том, что игра закночилась
38      */
39     private GameOverAnalyzer analyzerOfGameOver;
40
41     /**
42      * Наблюдатель за открытием дверей
43      * Оповещает слушателей о том, что была открыта дверь
44      */
45     private OpenDoorAnalyzer analyzerOfOpenDoor;
46
47     /**
48      * Наблюдатель за нахождением ключей
49      * Оповещает слушателей о том, что найден ключ
50      */
51     private FoundKeyAnalyzer analyzerOfFoundKey;
52
53     /**
54      * Анализатор пути
55      * Строит путь определённой длинны между двумя точкам
```

```
56          * Длинна ограниченна, потому что не хотелось бы, чтобы игрок в одно касание перешёл от начал к
        ↪ концу
57          */
58         private WayAnalyzer analyzerOfWay;
59
60         /**
61          * Массив дверей
62          * Запоминается, тк.. многократно используется при проверке возможности хода
63          */
64         private List<PointOnTheField> doors;
65
66
67         @Override
68         public void setGameField(final String newField) throws FieldIsEmptyException {
69
70             this.gameField = new GameField(newField);
71             protagonist = new Protagonist(gameField.getStartingPoint());
72             analyzerOfGameOver = new GameOverAnalyzer(this);
73             analyzerOfFoundKey = new FoundKeyAnalyzer(this);
74             analyzerOfOpenDoor = new OpenDoorAnalyzer(this);
75             analyzerOfWay = new WayAnalyzer(this);
76             doors = gameField.getDoors();
77
78             analyzerOfFoundKey.addListener(protagonist);
79             analyzerOfFoundKey.addListener(gameField);
80         }
81
82         /**
83          * Вызывается перед тем, как сделать ход, тк.. возможно дверь откроется
84          * @param point – точка, в которую попытался сходить игрок
85          */
86         private void checkDoors(PointOnTheField point) {
87
88             if (doors.contains(point) && protagonist.getNumberOfKeys() > 0) {
89
90                 protagonist.useKey();
91                 gameField.openDoor(point.getX(), point.getY());
92                 analyzerOfOpenDoor.doorIsOpen(point);
93             }
94         }
95
96         @Override
97         @org.jetbrains.annotations.Nullable
98         public ArrayDeque<PointOnTheField> movesOfProtagonist(final int x, final  int y) {
99
100            checkDoors(new PointOnTheField(x,y));
101
102            ArrayDeque<PointOnTheField> way = analyzerOfWay.getWay(getPositionOfProtagonist(), new
        ↪  PointOnTheField(x,y));
103
104            if (way == null) return null;
105
106            protagonist.movesOfProtagonist(x,y);
107
108            noticeAfterMotion();
109
110            return way;
111        }
112
113        /**
114         * Метод, который сообщает необходимым классам, что состояние системы изменилось
115         * Вызывается после хода игрока
116         */
117        private void noticeAfterMotion() {
118
119            analyzerOfGameOver.messageAboutChangingSystem();
120            analyzerOfFoundKey.messageAboutChangingSystem();
121        }
122
123        @Override
124        public boolean isItPassableCells(final int x, final int y) {
125
126            try {
127
128                return gameField.isItPassableCell(x,y);
129
```

```
130         }catch (OutOfBoundaryOfTheFieldException ex) {
131
132             return false;
133         }
134     }
135
136     @Override
137     public boolean isItPassableCells(final PointOnTheField point) {
138
139         try {
140
141             return gameField.isItPassableCell(point);
142
143         }catch (OutOfBoundaryOfTheFieldException ex) {
144
145             return false;
146         }
147     }
148
149     @Override
150     public int getSizeOfFieldX() {
151
152         return gameField.getSizeX();
153     }
154
155     @Override
156     public int getSizeOfFieldY() {
157
158         return gameField.getSizeY();
159     }
160
161     @Override
162     public List<PointOnTheField> getPassableCells() {
163
164         return gameField.getPassableCells();
165     }
166
167     @Override
168     public PointOnTheField getPositionOfProtagonist() {
169
170         return protagonist.getLocationOfProtagonist();
171     }
172
173     @Override
174     public PointOnTheField getStartPosition() {
175
176         return gameField.getStartingPoint();
177     }
178
179     @Override
180     public PointOnTheField getFinishPosition() {
181
182         return gameField.getFinishingPoint();
183     }
184
185     @Override
186     public void setValueOfRangeOfStep(int valueOfRangeOfStep) {
187
188         analyzerOfWay.setDefaultRange(valueOfRangeOfStep);
189     }
190
191     @Override
192     public List<PointOnTheField> getKeys() {
193
194         return gameField.getKeys();
195     }
196
197     @Override
198     public List<PointOnTheField> getDoors() {
199
200         return gameField.getDoors();
201     }
202
203
204     @Override
205     public void addListenerOfGameOver(GameOverListener listener) {
```

```
206
207         analyzerOfGameOver.addListener(listener);
208     }
209
210     @Override
211     public void removeListenerOfGameOver(GameOverListener listener) {
212
213         analyzerOfGameOver.removeListener(listener);
214     }
215
216     @Override
217     public void addListenerOfFoundKey(FoundKeyListener listener) {
218
219         analyzerOfFoundKey.addListener(listener);
220     }
221
222     @Override
223     public void removeListenerOfFoundKey(FoundKeyListener listener) {
224
225         analyzerOfFoundKey.removeListener(listener);
226     }
227
228     @Override
229     public void addListenerOfOpenDoor(OpenDoorListener listener) {
230
231         analyzerOfOpenDoor.addListener(listener);
232     }
233
234     @Override
235     public void removeListenerOfOpenDoor(OpenDoorListener listener) {
236
237         analyzerOfOpenDoor.removeListener(listener);
238     }
239
240     @Override
241     public int getNumberOfKeys() {
242
243         return protagonist.getNumberOfKeys();
244     }
245 }
```

```
1  package com.maltsev.labyrinth.model.protagonist;
2
3
4  import com.maltsev.labyrinth.model.analyzer.event.keysanddoors.keys.FoundKeyListener;
5  import com.maltsev.labyrinth.model.field.PointOnTheField;
6
7  /**
8   *  Главный герой
9   */
10 public class Protagonist implements FoundKeyListener {
11
12     /**
13      *  Позиция главного героя на карте
14      */
15     private PointOnTheField locationOfProtagonist;
16
17     /**
18      * Количество ключей, которые имеются у протагониста
19      */
20     private int numberOfKeys = 0;
21
22     @Override
23     public void keyIsFound(PointOnTheField positionOfKey) {
24
25         numberOfKeys++;
26     }
27
28     /**
29      * Ключ используется, следовательно количество ключей умменьшается
30      */
31     public void useKey() {
32
33         numberOfKeys--;
34     }
35
```

```
36      /**
37       * @return количество ключей на данный момент
38       */
39      public int getNumberOfKeys() {
40
41          return numberOfKeys;
42      }
43
44      /**
45       *  Перемещения протагониста на новую позицию
46       * @param newPoint – точка – новая позиция расположения героя
47       */
48      public void movesOfProtagonist(final PointOnTheField newPoint) {
49
50          locationOfProtagonist = new PointOnTheField(newPoint);
51      }
52
53      /**
54       *  Перемещения протагониста на новую позицию
55       * @param x координата точки по оси X, в которую нужно переместить протагониста
56       * @param y координата точки по оси Y, в которую нужно переместить протагониста
57       */
58      // уж если создал класс, представляющий позицию на поле, то его только и используй. Так что этот
        ↪ метод мне кажется лишним
59      //TODO иногда удобнее одно, иногда другое, по смыслу и то, и то допустимо
60      public void movesOfProtagonist(final int x, final  int y) {
61
62          locationOfProtagonist = new PointOnTheField(x,y);
63      }
64
65      /**
66       * Конструктор, с установкой начального положения героя
67       * @param startPoint точка – начальное положение героя
68       */
69      public Protagonist(final PointOnTheField startPoint) {
70
71          locationOfProtagonist = new PointOnTheField(startPoint);
72      }
73
74      /**
75       * Конструктор, с установкой начального положения героя
76       * @param x координата точки по оси X
77       * @param y координата точки по оси Y
78       */
79      Этот// тоже лишний
80      //TODO мне не сложно написать два варианта, они вполне допустимы, ты можешь сказать о позиции
        ↪ двумя параметрами или точкой, там где это используется улучшается понятность
81      public Protagonist(final int x, final  int y) {
82
83          locationOfProtagonist = new PointOnTheField(x, y);
84      }
85
86      /**
87       * @return точка, положения героя
88       */
89      public PointOnTheField getLocationOfProtagonist() {
90
91          return locationOfProtagonist;
92      }
93  }
```

```
1   package com.maltsev.labyrinth.model.field;
2
3   /**
4    * Точка на поле
5    */
6   public class PointOnTheField {
7
8       /**
9        *  Координата точки по оси X
10       */
11      private int x;
12
13      /**
14       *  Координата точки по оси Y
15       */
```

```
16      private int y;
17
18
19      /**
20       * Конструктор точки
21       * @param x координата по оси X
22       * @param y координата по оси Y
23       */
24      public PointOnTheField(final int x, final int y) {
25
26          this.x = x;
27          this.y = y;
28      }
29
30      /**
31       *  Конструктор копирования
32       * @param point – объект, который копируют
33       */
34      public PointOnTheField(final PointOnTheField point) {
35
36          this.x = point.x;
37          this.y = point.y;
38      }
39
40      /**
41       * @return координату по оси X
42       */
43      public int getX() {
44
45          return x;
46      }
47
48      /**
49       * @return координату по оси Y
50       */
51      public int getY() {
52
53          return y;
54      }
55
56      @Override
57      public boolean equals(final Object obj) {
58
59          if (obj == this) {
60
61              return true;
62          }
63
64          if (obj == null || obj.getClass() != this.getClass()) {
65
66              return false;
67          }
68
69          PointOnTheField comparePoint = (PointOnTheField) obj;
70
71          return this.x == comparePoint.x && this.y == comparePoint.y;
72      }
73
74      /**
75       * Сгенерировала IDEA
76       */
77      @Override
78      public int hashCode() {
79
80          int result = x;
81          result = 31 * result + y;
82          return result;
83      }
84
85      /**
86       * @param x координата точки по оси X
87       * @param y координата точки по оси Y
88       * @return одинаковые ли точки
89       */
90      public boolean equals(final int x, int y) {
91
```

```
92          return this.x == x && this.y == y;
93      }
94
95      /**
96       * Сгенерировала IDEA
97       */
98      @Override
99      public String toString() {
100
101         return "PointOnTheField{" +
102                 "x=" + x +
103                 ", y=" + y +
104                 '}';
105     }
106 }
```

```
1  package com.maltsev.labyrinth.model.field;
2
3  /**
4   *  Выход за границу поля - исключение
5   */
6  public class OutOfBoundaryOfTheFieldException extends Exception{
7
8      /**
9       *  Имя параметра, который не соответствует норме
10      */
11     private String nameOfParam;
12
13     /**
14      *  Значение параметра, которое привело к исключению
15      */
16     private int valueOfParam;
17
18     /**
19      *  Максимально допустимое значение параметра
20      */
21     private int maximumAllowableValueOfParam;
22
23     /**
24      * Имя класса, который бросил исключение
25      */
26     private String infoAboutException;
27
28     /**
29      * Конструктор, в который передаётся информация об ошибке
30      * @param infoAboutException информация, о брошенном исклчюение
31      * @param nameOfParam имя параметра, который не соответствует норме
32      * @param valueOfParam значение параметра, которое привело к исключению
33      * @param maximumAllowableValueOfParam максимально допустимое значение параметра
34      */
35     public OutOfBoundaryOfTheFieldException(final String infoAboutException, final String
    ↪ nameOfParam,
36                                             final int valueOfParam, final int
    ↪ maximumAllowableValueOfParam) {
37
38         super("\n\n" + infoAboutException + " :" +
39                 "\Неправильнон задан " + nameOfParam + " : " + valueOfParam +
40                 "\Допустимоен значение : [0;" + maximumAllowableValueOfParam + "]");
41
42         this.nameOfParam = nameOfParam;
43         this.valueOfParam = valueOfParam;
44         this.maximumAllowableValueOfParam = maximumAllowableValueOfParam;
45         this.infoAboutException = infoAboutException;
46     }
47
48     /**
49      * @return имя параметра, который не соответствует норме
50      */
51     public String getNameOfParam() {
52
53         return nameOfParam;
54     }
55
56     /**
57      * @return значение параметра, которое привело к исключению
58      */
```

```
59      public int getValueOfParam() {
60
61          return valueOfParam;
62      }
63
64      /**
65       * @return максимально допустимое значение параметра
66       */
67      public int getMaximumAllowableValueOfParam() {
68
69          return maximumAllowableValueOfParam;
70      }
71
72      /**
73       * @return откуда исключение прилетело
74       */
75      public String getInfoAboutException() {
76
77          return infoAboutException;
78      }
79  }
```

```
1   package com.maltsev.labyrinth.model.field;
2
3   import com.maltsev.labyrinth.model.analyzer.event.keysanddoors.keys.FoundKeyListener;
4
5   import java.util.ArrayList;
6   import java.util.List;
7
8   /**
9    *  Игровое поле
10   */
11  public class GameField implements FoundKeyListener{
12
13      /**
14       *  Матрица ячеек, те.. само поле
15       */
16      private CellOfField[][] field;
17
18      /**
19       *  Массив координат ячеек, которые являются проходимыми
20       */
21      private List<PointOnTheField> passableCells;
22
23      /**
24       *  Массив координат ячеек, в которых находятся двери
25       */
26      private List<PointOnTheField> doors;
27
28      /**
29       *  Массив координат ячеек, в которых находятся ключи
30       */
31      private List<PointOnTheField> keys;
32
33      /**
34       *  Размер поля по оси X
35       */
36      private int sizeOfFieldX;
37
38      /**
39       *  Размер поля по оси Y
40       */
41      private int sizeOfFieldY;
42
43      /**
44       * Стартовая точка поля
45       */
46      private PointOnTheField startingPoint;
47
48      /**
49       * Финишная точка поля
50       */
51      private PointOnTheField finishingPoint;
52
53
54      /**
```

```
55          * Конструктор, в которм создаётся игровое поле
56          * @param newField строкаматрица-, где 1,s,f - проходимые элементы, а 0 - нет,
57          *                 s - начальная точка поля, f - конечная, а новая строчка задаётся \n
58          */
59         public GameField(final String newField) throws FieldIsEmptyException{
60
61             passableCells = new ArrayList<PointOnTheField>();
62             doors = new ArrayList<PointOnTheField>();
63             keys = new ArrayList<PointOnTheField>();
64
65             String[] fieldFromString = newField.split("\\n");    // делим полученую строчку на части
66
67             int lengthX = fieldFromString.length;
68             int lengthY = fieldFromString[fieldFromString.length - 1].length();
69
70
71             for(int x = 0; x < lengthX; x++) {                          // Проверка на то, чтобы
    ↪ матрица была прямоугольной,
72                                                                         // если это не так, то шириной
    ↪ берётся минимальное значение
73                 if (lengthY > fieldFromString[x].length()) {            // остальное отбрасывается
74
75                     lengthY = fieldFromString[x].length();
76                 }
77             }
78
79             field = new CellOfField[lengthX][lengthY];
80
81             boolean metBeginning = false;
82             boolean metEnd = false;
83
84             for(int x = 0; x < lengthX; x++) {
85
86                 for(int y = 0; y < lengthY; y++) {
87
88                     boolean isItPossibleWay = false;
89
90                     if (fieldFromString[x].charAt(y) != '0') {              // Установка проходимости
91
92                         isItPossibleWay = true;
93
94                         passableCells.add(new PointOnTheField(x,y));
95                     }
96
97                     field[x][y] = new CellOfField(isItPossibleWay);
98
99                     switch (fieldFromString[x].charAt(y)) {
100
101                        case 's':            // Установка стартовой и финишной точек
102                        case 'S': {
103
104                            if (!metBeginning) startingPoint = new PointOnTheField(x,y);
105                            metBeginning = true;
106                            break;
107                        }
108                        case 'f':
109                        case 'F': {
110
111                            if (!metEnd) finishingPoint = new PointOnTheField(x,y);
112                            metEnd = true;
113                            break;
114                        }
115
116                        case 'd':            // Устновка дверей и ключей
117                        case 'D': {
118
119                            doors.add(new PointOnTheField(x,y));
120                            field[x][y].createDoor();
121                            break;
122                        }
123                        case 'k':
124                        case 'K': {
125
126                            keys.add(new PointOnTheField(x,y));
127                            break;
128                        }
```

```
129                 }
130
131             }
132         }
133
134         setSize();
135
136         if(passableCells.size() < 2)
137             throw new FieldIsEmptyException("This field is empty dude");
138
139         if (!metBeginning) {                                        // Если точки начала и
    ↪ конца не обнаружены, то
140                                                                     //   они выбираются, из
    ↪ проходимых ячеек,
141             PointOnTheField startPoint = passableCells.get(0);      //   как первая и
    ↪ последняя
142             this.startingPoint = new PointOnTheField(startPoint);
143         }
144         if (!metEnd) {
145
146             PointOnTheField finishPoint = passableCells.get(passableCells.size() - 1);
147             this.finishingPoint = new PointOnTheField(finishPoint);
148         }
149     }
150
151     /**
152      *  Фиксирование размеров поля
153      */
154     private void setSize() {
155
156         sizeOfFieldX = field.length;
157         sizeOfFieldY = field[0].length;
158     }
159
160     /**
161      * @param x - координата точки по оси X
162      * @param y - координата точки по оси Y
163      * @return является ли эта ячейка проходимой
164      * @throws OutOfBoundaryOfTheFieldException - воход за грациуц поля
165      */
166     public boolean isItPassableCell(final int x, final int y) throws
    ↪ OutOfBoundaryOfTheFieldException {
167
168         if (x < 0 || x >= sizeOfFieldX)
169             throw new OutOfBoundaryOfTheFieldException("Illegal request of possible way",
170                     "x", x, sizeOfFieldX - 1);
171
172         if (y < 0 || y >= sizeOfFieldY)
173             throw new OutOfBoundaryOfTheFieldException("Illegal request of possible way",
174                     "y", y, sizeOfFieldY - 1);
175
176         return field[x][y].getInfoAboutPatencyOfCell();
177     }
178
179     /**
180      * @param point точка на поле указывающая на ячейку
181      * @return является ли эта ячейка проходимой
182      * @throws OutOfBoundaryOfTheFieldException - воход за грациуц поля
183      */
184     public boolean isItPassableCell(final PointOnTheField point) throws
    ↪ OutOfBoundaryOfTheFieldException {
185
186         int x = point.getX();
187         int y = point.getY();
188
189         if (x < 0 || x >= sizeOfFieldX)
190             throw new OutOfBoundaryOfTheFieldException("Illegal request of possible way",
191                     "x", x, sizeOfFieldX - 1);
192
193         if (y < 0 || y >= sizeOfFieldY)
194             throw new OutOfBoundaryOfTheFieldException("Illegal request of possible way",
195                     "y", y, sizeOfFieldY - 1);
196
197         return field[x][y].getInfoAboutPatencyOfCell();
198     }
199
```

```
200     /**
201      * @return размер поля по оси X начиная( отсчёт с нуля)
202      */
203     public int getSizeX() {
204 
205         return sizeOfFieldX;
206     }
207 
208     /**
209      * @return размер поля по оси Y начиная( отсчёт с нуля)
210      */
211     public int getSizeY() {
212 
213         return sizeOfFieldY;
214     }
215 
216     /**
217      * @return начальная точка поля
218      */
219     public PointOnTheField getStartingPoint() {
220 
221         return startingPoint;
222     }
223 
224     /**
225      * @return конечная точка поля
226      */
227     public PointOnTheField getFinishingPoint() {
228 
229         return finishingPoint;
230     }
231 
232     /**
233      * @return Массив координат проходимых ячеек
234      */
235     public List<PointOnTheField> getPassableCells() {
236 
237         return new ArrayList<PointOnTheField>(passableCells);
238     }
239 
240     /**
241      * @return Массив координат расположения дверей
242      */
243     public List<PointOnTheField> getDoors() {
244 
245         return new ArrayList<PointOnTheField>(doors);
246     }
247 
248     /**
249      * @return Массив координат расположения ключей
250      */
251     public List<PointOnTheField> getKeys() {
252 
253         return new ArrayList<PointOnTheField>(keys);
254     }
255 
256     /**
257      * Открыть дверь
258      * @param x координата по оси X
259      * @param y координата по оси Y
260      */
261     public void openDoor(int x, int y) {
262 
263         field[x][y].openDoor();
264     }
265 
266     @Override
267     public void keyIsFound(PointOnTheField keyPosition) {
268 
269         keys.remove(keyPosition);
270     }
271 }
```

```
1 package com.maltsev.labyrinth.model.field;
2 
3 /**
```

```
 * Исключение, которое бросается, если поле пустое и ходить некуда
 */
public class FieldIsEmptyException extends Exception{

    FieldIsEmptyException(String str) {

        super(str);
    }
}
```

```
package com.maltsev.labyrinth.model.field;Странный

// класс. По сути же это boolean
//TODO это сделано для расширяемости

/**
 *  Ячейка поля, содержит информацию, о том возможно ли на неё поместить протагониста или нет
 */
class CellOfField {

    /**
     * Поле хранящее информацию о проходимости клетки
     */
    private boolean isThisCellPossibleForMove;

    /**
     * Есть ли дверь в клетке
     */
    private boolean isDoor = false;

    /**
     * Открыта ли дверь
     */
    private boolean doorIsOpen = false;

    /**
     * @return проходима ли клетка
     */
    boolean getInfoAboutPatencyOfCell() {

        if(!isDoor)
            return isThisCellPossibleForMove;
        else
            return doorIsOpen;
    }

    /**
     * @param isThisCellPossibleForMove является ли эта ячейка проходимой
     */
    CellOfField(final boolean isThisCellPossibleForMove) {

        this.isThisCellPossibleForMove = isThisCellPossibleForMove;
    }

    /**
     * Установить дверь в клетку
     */
    void createDoor() {

        if(isThisCellPossibleForMove)
            isDoor = true;
    }

    /**
     * Открытие двери
     */
    void openDoor() {

        if(isDoor)
            doorIsOpen = true;
    }
}
```

```
package com.maltsev.labyrinth.model.analyzer;

```

```
3
4  import com.maltsev.labyrinth.model.Model;
5  import com.maltsev.labyrinth.model.field.PointOnTheField;
6
7  import java.util.ArrayDeque;
8
9  /**
10  * Класс, главная задача которого проведение маршрута из одной точки в другую
11  */
12 public class WayAnalyzer {
13
14     private Model model;
15
16     private int[][] fieldForWave;
17
18     private int defaultRange = 5;
19
20     private int range;
21
22     private PointOnTheField startPoint;                // По факту эти поля вынесены сюда, чтобы
       ↪ было удобнее делигировать
23     private PointOnTheField finishPoint;              //  выолнение различных частей алгоритма
       ↪ другим методам
24
25     private int sizeOfFieldX;
26     private int sizeOfFieldY;
27
28     public WayAnalyzer(Model model) {
29
30         this.model = model;
31     }
32
33     /**
34      * Установить значение для длины шага
35      * @param defaultRange значение, которое вы хотели бы устновить
36      */
37     public void setDefaultRange(int defaultRange) {
38
39         this.defaultRange = defaultRange;
40     }
41
42     /**
43      * Может быть это комуто– интересно
44      * @return дефолтная длина шага
45      */
46     public int getDefaultRange() {
47
48         return defaultRange;
49     }
50
51     /**
52      * Метод по умолчанию, вызывает метод поиска пути с дефолтной длинной шага
53      */
54     @org.jetbrains.annotations.Nullable
55     public ArrayDeque<PointOnTheField> getWay(final PointOnTheField startPoint, final
       ↪ PointOnTheField finishPoint) {
56
57         return getWay(startPoint, finishPoint, defaultRange);
58     }
59
60     /**
61      * Построение пути из одной точки поля в другую
62      * @param startPoint стартовая точка пути
63      * @param finishPoint финишная точка пути
64      * @return возвращяет либо путь в виде массива точек, либо ноль, если путь содержит более range
       ↪ шагов или он невозможен
65      */
66     @org.jetbrains.annotations.Nullable
67     public ArrayDeque<PointOnTheField> getWay(final PointOnTheField startPoint, final
       ↪ PointOnTheField finishPoint, int range) {
68
69         if (!model.isItPassableCells(startPoint) || !model.isItPassableCells(finishPoint))
70             return null;                                // Проверка на правильность задания начальной
       ↪ и конечной точки
71
72         this.startPoint = startPoint;
```

```
73          this.finishPoint = finishPoint;                 // Записываем введённые параметры
74          this.range = range;
75
76          sizeOfFieldX = model.getSizeOfFieldX();         // Выписываем из модели значения размера поля
77          sizeOfFieldY = model.getSizeOfFieldY();
78
79          fieldForWave = new int[sizeOfFieldX][sizeOfFieldY];
80
81          putMaximumValueInEveryCell();
82
83          if(!motionOfWave()) return null;
84
85          //ArrayList<PointOnTheField> wayBack = returningWave();
86
87          ArrayDeque<PointOnTheField> way = returningWave();
88
89          if (way == null) return null;
90
91          return way;
92      }
93
94      /**
95       * Возвращение волны
96       * @return либо путь возвращения в виде коллекции точек, либо null
97       */
98      @org.jetbrains.annotations.Nullable
99      private ArrayDeque<PointOnTheField> returningWave() {
100
101         ArrayDeque<PointOnTheField> way = new ArrayDeque<PointOnTheField>();
102
103         PointOnTheField bufferPoint = new PointOnTheField(finishPoint);
104
105         way.addFirst(bufferPoint);
106
107         int weightOfTheWave;
108         int noMoreThanFewStrokes = 0;
109
110         while (!bufferPoint.equals(startPoint)) {
111
112             int bufferPointX = bufferPoint.getX();
113             int bufferPointY = bufferPoint.getY();
114             weightOfTheWave = fieldForWave[bufferPointX][bufferPointY];
115
116             if (bufferPointX + 1 < sizeOfFieldX && fieldForWave[bufferPointX + 1][bufferPointY
    ↪ ] == weightOfTheWave - 1) {
117
118                 bufferPoint = new PointOnTheField(bufferPointX + 1, bufferPointY);
119                 way.addFirst(bufferPoint);
120             } else
121             if (bufferPointY + 1 < sizeOfFieldY && fieldForWave[bufferPointX][bufferPointY +
    ↪ 1] == weightOfTheWave - 1) {
122
123                 bufferPoint = new PointOnTheField(bufferPointX, bufferPointY + 1);
124                 way.addFirst(bufferPoint);
125             } else
126             if (bufferPointX - 1 >= 0 && fieldForWave[bufferPointX - 1][bufferPointY] ==
    ↪ weightOfTheWave - 1) {
127
128                 bufferPoint = new PointOnTheField(bufferPointX - 1, bufferPointY);
129                 way.addFirst(bufferPoint);
130             } else
131             if (bufferPointY - 1 >= 0 && fieldForWave[bufferPointX][bufferPointY - 1] ==
    ↪ weightOfTheWave - 1) {
132
133                 bufferPoint = new PointOnTheField(bufferPointX, bufferPointY - 1);
134                 way.addFirst(bufferPoint);
135             }
136
137             noMoreThanFewStrokes++;
138
139             if (noMoreThanFewStrokes > range)  return null;
140
141         } // проходим от конца к началу, запоминая путь
142
143         return way;
144     }
```

```
145
146     /**
147      * Распространение волны: в цикле из начальной точки выходит волна и заполняет всё поле
148      */
149     private boolean motionOfWave() {
150
151         fieldForWave[startPoint.getX()][startPoint.getY()] = 0;      // стартовая точка волны
152         fieldForWave[finishPoint.getX()][finishPoint.getY()] = Integer.MAX_VALUE;   // конечная
    ↪ точка волны
153
154         int weightOfTheWave = 0;
155
156         int noMoreThanFewStrokes = 0;
157
158         while (fieldForWave[finishPoint.getX()][finishPoint.getY()] == Integer.MAX_VALUE) {
159
160             for (int i = 0; i < fieldForWave.length; i++) {
161
162                 for (int j = 0; j < fieldForWave[i].length; j++) {
163
164                     if (fieldForWave[i][j] == weightOfTheWave) {
165
166                         if (i + 1 < sizeOfFieldX && fieldForWave[i + 1][j] > weightOfTheWave
    ↪ && model.isItPassableCells(i + 1, j)) {
167
168                             fieldForWave[i + 1][j] = weightOfTheWave + 1;
169                         }
170                         if (j + 1 < sizeOfFieldY && fieldForWave[i][j + 1] > weightOfTheWave
    ↪ && model.isItPassableCells(i, j + 1)) {
171
172                             fieldForWave[i][j + 1] = weightOfTheWave + 1;
173                         }
174                         if (i - 1 >= 0 && fieldForWave[i - 1][j] > weightOfTheWave  && model.
    ↪ isItPassableCells(i - 1, j)) {
175
176                             fieldForWave[i - 1][j] = weightOfTheWave + 1;
177                         }
178                         if (j - 1 >= 0 && fieldForWave[i][j - 1] > weightOfTheWave && model.
    ↪ isItPassableCells(i, j - 1)) {
179
180                             fieldForWave[i][j - 1] = weightOfTheWave + 1;
181                         }
182                     }
183                 }
184             }
185
186             weightOfTheWave++;
187
188             noMoreThanFewStrokes++;
189             if (noMoreThanFewStrokes > range)  return false;
190         }
191
192         return true;
193     }
194
195     /**
196      * Заполнение поля максимальными значениями
197      */
198     private void putMaximumValueInEveryCell() {
199
200         for (int i = 0; i < fieldForWave.length; i++) {
201
202             for (int j = 0; j < fieldForWave[i].length; j++) {
203
204                 fieldForWave[i][j] = Integer.MAX_VALUE;
205             }
206         }
207     }
208 }
```

```
1 package com.maltsev.labyrinth.model.analyzer.event;
2
3
4 import com.maltsev.labyrinth.model.Model;
5
6 /**
```

```
 * Обобщающий класс для тех, кто следит за событиями и оповещает других, если оно произошло
 */
public abstract class EventAnalyzer {

    protected Model model;

    protected EventAnalyzer(Model model) {

        this.model = model;
    }

}
```

```
package com.maltsev.labyrinth.model.analyzer.event.gameover;Мне

// кажется довольно странным, что GameOverAnalyzer содержит поле ModelOfLabyrinth, a ModelOfLabyrinth содержит поле GameOverAnalyzer
//todo это вполне нормально, ведь GameOverAnalyzer должен получать информацию про игровое поле и ему не сильно интересно, что там внутри, следовательно ему удобнее работать с  ModelOfLabyrinth

import com.maltsev.labyrinth.model.Model;
import com.maltsev.labyrinth.model.analyzer.event.EventAnalyzer;
import com.maltsev.labyrinth.model.field.PointOnTheField;

import java.util.LinkedList;
import java.util.PriorityQueue;
import java.util.Queue;

public class GameOverAnalyzer extends EventAnalyzer {

    private PointOnTheField finishPoint;

    private Queue<GameOverListener> queue;



    public GameOverAnalyzer(Model model) {

        super(model);

        queue = new LinkedList<GameOverListener>();

        finishPoint = new PointOnTheField(model.getFinishPosition());
    }

    /**
     * Добавить слушателя
     * @param listener объектслушатель–
     */
    public void addListener(GameOverListener listener) {

        queue.add(listener);
    }

    /**
     * Используется, чтобы отписаться от рассылки
     * @param listener – объектслушатель–
     */
    public void removeListener(GameOverListener listener) {

        if (queue.contains(listener))
            queue.remove(listener);
    }


    /**
     * Используется, когда состояние системы изменилось
     */
    public void messageAboutChangingSystem() {

        if (finishPoint.equals(model.getPositionOfProtagonist())) {

            alertListener();
        }
    }

```

```
62      /**
63       * Оповещание слушателей об окончание игры
64       */
65      private void alertListener() {
66
67          GameOverListener item;
68
69          while (!queue.isEmpty()) {
70
71              item = queue.poll();
72              item.gameIsOver();
73          }
74      }
75  }
```

```
1  package com.maltsev.labyrinth.model.analyzer.event.gameover;
2
3
4  /**
5   * Интерфейс для записи в слушатели информации о прохождение поля
6   */
7  public interface GameOverListener {
8
9      /**
10      * Метод, который вызовется в случае окончания игры
11      */
12     void gameIsOver();
13 }
```

```
1  package com.maltsev.labyrinth.model.analyzer.event.keysanddoors.doors;
2
3
4  import com.maltsev.labyrinth.model.Model;
5  import com.maltsev.labyrinth.model.analyzer.event.EventAnalyzer;
6  import com.maltsev.labyrinth.model.field.PointOnTheField;
7
8  import java.util.*;
9
10 /**
11  * Извещает интересующихся об открытие двери
12  */
13 public class OpenDoorAnalyzer extends EventAnalyzer{
14
15     private Queue<OpenDoorListener> queue;
16
17     List<PointOnTheField> doors;
18
19
20     public OpenDoorAnalyzer(Model model) {
21
22         super(model);
23
24         queue = new LinkedList<OpenDoorListener>();
25
26         doors = model.getDoors();
27     }
28
29     /**
30      * Добавить слушателя
31      * @param listener объектслушатель–
32      */
33     public void addListener(OpenDoorListener listener) {
34
35         queue.add(listener);
36     }
37
38     /**
39      * Используется, чтобы отписаться от рассылки
40      * @param listener – кого удалить
41      */
42     public void removeListener(OpenDoorListener listener) {
43
44         if (queue.contains(listener))
45             queue.remove(listener);
46     }
```

```
47
48      /**
49       * Вызывается при открытие двери, чтобы оповестить об этом слушателей
50       */
51      public void doorIsOpen(PointOnTheField doorPosition) {
52
53          alertListener(doorPosition);
54      }
55
56      /**
57       * Оповещание слушателей об окончание игры
58       */
59      private void alertListener(PointOnTheField doorPosition) {
60
61          OpenDoorListener item;
62
63          for (int i = 0; i < queue.size(); i++) {
64
65              item = queue.poll();
66              item.doorIsOpen(doorPosition);
67              queue.add(item);
68          }
69      }
70  }
```

```
1  package com.maltsev.labyrinth.model.analyzer.event.keysanddoors.doors;
2
3
4  import com.maltsev.labyrinth.model.field.PointOnTheField;
5
6  /**
7   * Для оповещения об открытие двери, реализуёте этот интерфейс
8   */
9  public interface OpenDoorListener {
10
11      /**
12       * Метод, который вызывется у всех слушателей, после открытия двери
13       */
14      void doorIsOpen(PointOnTheField doorPosition);
15  }
```

```
1  package com.maltsev.labyrinth.model.analyzer.event.keysanddoors.keys;
2
3
4  import com.maltsev.labyrinth.model.Model;
5  import com.maltsev.labyrinth.model.analyzer.event.EventAnalyzer;
6  import com.maltsev.labyrinth.model.field.PointOnTheField;
7
8  import java.util.LinkedList;
9  import java.util.List;
10 import java.util.Queue;
11
12 /**
13  * Класс, который следит за получением ключей
14  */
15 public class FoundKeyAnalyzer extends EventAnalyzer {
16
17     private Queue<FoundKeyListener> queue;
18
19     private List<PointOnTheField> keys;
20
21     private PointOnTheField keyPosition;
22
23
24     public FoundKeyAnalyzer(Model model) {
25
26         super(model);
27
28         queue = new LinkedList<FoundKeyListener>();
29
30         keys = model.getKeys();
31     }
32
33     /**
34      * Добавить слушателя
```

```
35      * @param listener объектслушатель–
36      */
37     public void addListener(FoundKeyListener listener) {
38 
39         queue.add(listener);
40     }
41 
42     /**
43      * Используется, чтобы отписаться от рассылки
44      * @param listener – кого удалить
45      */
46     public void removeListener(FoundKeyListener listener) {
47 
48         if (queue.contains(listener))
49             queue.remove(listener);
50     }
51 
52     /**
53      * Сообщение о том, что состояние системы изменилось и возможно
54      * настал момент истины тот( момент, ради которго создан класс)
55      */
56     public void messageAboutChangingSystem() {
57 
58         keyPosition = model.getPositionOfProtagonist();     Не// всегда на этой позиции есть ключь,
    ↪  но мы его там ожидаем
59 
60         if(keys.contains(keyPosition))
61             alertListener();
62     }
63 
64     /**
65      * Оповещание слушателей об окончание игры
66      */
67     private void alertListener() {
68 
69         FoundKeyListener item;
70 
71         for (int i = 0; i < queue.size(); i++) {
72 
73             item = queue.poll();
74             item.keyIsFound(keyPosition);          // Здесь keyPosition уже всегда позиция ключа на
    ↪ карте
75             queue.add(item);
76         }
77 
78         keys = model.getKeys();
79     }
80 
81 }
```

```
1  package com.maltsev.labyrinth.model.analyzer.event.keysanddoors.keys;
2 
3 
4  import com.maltsev.labyrinth.model.field.PointOnTheField;
5 
6  /**
7   * Хочешь получить сообщение о получение ключа – реализуй интерфейс и вставь в очередьбез( интерфейса не
       ↪ принимаем)
8   */
9  public interface FoundKeyListener {
10 
11     /**
12      * Метод, вызывающийся у слушателей, после обнаружения ключа
13      * @param keyPosition позиция ключа
14      */
15     void keyIsFound(PointOnTheField keyPosition);
16 }
```

```
1  package com.maltsev.labyrinth.presenter;
2 
3  import com.maltsev.labyrinth.model.Model;
4  import com.maltsev.labyrinth.model.ModelOfLabyrinth;
5  import com.maltsev.labyrinth.model.analyzer.event.gameover.GameOverListener;
6  import com.maltsev.labyrinth.model.analyzer.event.keysanddoors.doors.OpenDoorListener;
7  import com.maltsev.labyrinth.model.analyzer.event.keysanddoors.keys.FoundKeyListener;
```

```
8  import com.maltsev.labyrinth.model.field.FieldIsEmptyException;
9  import com.maltsev.labyrinth.model.field.PointOnTheField;
10 import com.maltsev.labyrinth.presenter.interfaces.View;
11 import com.maltsev.labyrinth.presenter.tempdata.PointOnTheScreen;
12 import com.maltsev.labyrinth.presenter.tempdata.SizeOfTexture;
13 import static com.maltsev.labyrinth.presenter.ParsingFile.getFieldOnTheNumber;
14
15 import java.util.ArrayDeque;
16 import java.util.ArrayList;
17 import java.util.List;
18
19 /**
20  * Presenter выступает в качестве посредника между View и ModelOfLabyrinth.
21  * Он извлекает данные из модели и передает их во View. Также решает,
22  * что нужно делать, когда вы взаимодействуете с View.
23  */
24 public class Presenter implements GameOverListener, FoundKeyListener, OpenDoorListener {
25
26     private Model model;
27     private View view;
28
29     private SizeOfTexture sizeOfBlock;
30     private List<PointOnTheField> passableCells;
31     private List<PointOnTheField> doorsClosed;
32     private List<PointOnTheField> doorsOpened;
33     private List<PointOnTheField> keys;
34
35     private ArrayDeque<PointOnTheField> way;
36
37     private PointOnTheScreen pointOfMovement;
38     private double timer = 0;
39     private double rateOfProtagonist = 0.2;
40
41     private int rangeOfStep = 7;
42
43     /**
44      * Аналогично isKeyFound, создано чтобы отрисовка поля происходила после того, как протагонист
       ↪ закончит движение
45      */
46     private boolean isGameOver = false;
47
48     /**
49      * Поле, созданное, чтобы ключ исчезал с карты после окончания движения протагониста, а не при
       ↪ нажатие на ключ
50      */
51     private boolean isKeyFound = false;
52
53
54     /**
55      * Следует вызвать последним, так как он интересуется полями View может( оказаться, что поля ещё не
       ↪ инициализированы)
56      * @param view отрисовщик ui, с которым будет работать presenter
57      */
58     public Presenter(View view, int numberOfTheField) {
59
60         this.view = view;
61
62         model = new ModelOfLabyrinth();
63
64
65         String newField = getFieldOnTheNumber(numberOfTheField);
66
67         try {
68
69             model.setGameField(newField);
70         }
71         catch (FieldIsEmptyException ex){
72
73             System.out.println(ex);
74
75             String defaultField = "11111\n11111";
76
77             try {
78                 model.setGameField(defaultField);
79             }
80             catch (FieldIsEmptyException error) {
```

```
81
82                  throw new Error("Всё очень плохо");
83              }
84          }
85
86          model.addListenerOfGameOver(this);
87          model.addListenerOfFoundKey(this);
88          model.addListenerOfOpenDoor(this);
89          model.setValueOfRangeOfStep(rangeOfStep);
90
91          sizeOfBlock = new SizeOfTexture(view.getSizeOfBlock());
92          passableCells = model.getPassableCells();
93          doorsClosed = model.getDoors();
94          doorsOpened = new ArrayList<PointOnTheField>();
95          keys = model.getKeys();
96      }
97
98      /**
99       * Отрисовывает поле
100      * Вызывает метод drawBlock() drawExit() drawKeys() drawClosedDoors() y View
101      */
102     public void drawField() {
103
104         drawPassableCells();
105         drawExit();
106
107         drawKeys();
108         drawClosedDoors();
109         drawOpenedDoors();
110     }
111
112     /**
113      * Отрисовка клеток, по которым можно передвигаться протагонисту
114      * Вызывает метод drawBlock() y View
115      */
116     public void drawPassableCells() {
117
118         for (PointOnTheField point : passableCells) {
119
120             view.drawBlock(translatePointFieldToScreen(point));
121         }
122     }
123
124     /**
125      * Отрисовывает финишную клетку
126      * Вызывает метод drawExit() y View
127      */
128     public void drawExit() {
129
130         view.drawExit(translatePointFieldToScreen(model.getFinishPosition()));
131     }
132
133     /**
134      * Отрисовывает ключи на карте
135      * Вызывает метод drawKey() y View
136      */
137     public void drawKeys() {
138
139         for (PointOnTheField point : keys) {
140
141             view.drawKey(translatePointFieldToScreen(point));
142         }
143     }
144
145     /**
146      * Отрисовывает закрытые двери на карте
147      * Вызывает метод drawCloseDoor() y View
148      */
149     public void drawClosedDoors() {
150
151         for (PointOnTheField point : doorsClosed) {
152
153
154             view.drawCloseDoor(translatePointFieldToScreen(point));
155         }
156     }
```

```
157
158     /**
159      * Отрисовывает двери на карте
160      * Вызывает метод drawOpenDoor() у View
161      */
162     public void drawOpenedDoors() {
163
164         for (PointOnTheField point : doorsOpened) {
165
166             view.drawOpenDoor(translatePointFieldToScreen(point));
167         }
168     }
169
170     /**
171      * Движение протагониста
172      * Если движение в указаную точку невозможно, то никаких действий совершенно не будет
173      * @param screenX координата экрана по оси X
174      * @param screenY координата экрана по оси Y
175      */
176     public void moveProtagonist(float screenX, float screenY) {
177
178         int x = (int) screenX / sizeOfBlock.getWidth();
179         int y = (int) screenY / sizeOfBlock.getHeight();
180
181         way = model.movesOfProtagonist(x, y);
182
183         if (way != null)
184             startMovement();
185     }
186
187     /**
188      * Метод, который вызывается в начале движения
189      * Вызывает методы lockInput() и startMovement() у View
190      */
191     private void startMovement() {
192
193         view.lockInput();
194         view.startMovement();
195         timer = 0;
196         pointOfMovement = translatePointFieldToScreen(way.poll());
197     }
198
199     /**
200      * Действия совершаемые, по окончаанию движения протагониста
201      * Например, открытие ввода
202      *
203      * Отрисовка изменений внесённых ходом
204      * Например, исчезновение ключа или сообщение об окончание игры
205      */
206     private void finishMovement() {
207
208         view.unlockInput();
209         view.finishMovement();
210         timer = 0;
211
212         if (isKeyFound) {
213
214             keys = model.getKeys();
215             isKeyFound = false;
216         }
217
218         if (isGameOver) {
219
220             view.lockInput();
221             view.messageOfGameOver();
222         }
223     }
224
225     /**
226      * Отрисовка движения протагониста
227      * По очереди отрисовываются все клетки пути между начальной точкой и конечной
228      * @param deltaTime сколько времени прошло относительно прошлой отрисовки, добавленно, чтобы
    ↪ регулировать скороть
229      *                  передвижения протагониста
230      */
231     private void movementOfProtagonist(float deltaTime) {
```

```
232
233             timer += deltaTime;
234
235             if(timer > rateOfProtagonist) {
236
237                 timer = 0;
238                 pointOfMovement = translatePointFieldToScreen(way.poll());
239             }
240
241
242             if(way.isEmpty())
243                 finishMovement();
244         }
245
246         /**
247          * Вызывается, чтобы узнать где находится протагонист во время передвижения
248          * @param deltaTime промежуток времени между кадрами
249          * @return Точка, в которой находится двигающийся протагонист в данный момент
250          */
251         public PointOnTheScreen getPositionOfMovingProtagonist(float deltaTime) {
252
253             movementOfProtagonist(deltaTime);
254
255             return pointOfMovement;
256         }
257
258         /**
259          * @return позиция протагониста, в координатах экрана
260          */
261         public PointOnTheScreen getPositionOfProtagonist() {
262
263             return  translatePointFieldToScreen(model.getPositionOfProtagonist());
264         }
265
266         /**
267          * Перевод координат точки из координат поля в координаты экрана
268          * @param pointOnTheField точка с координатами поля
269          * @return точка с координатами экрана
270          */
271         private PointOnTheScreen translatePointFieldToScreen(PointOnTheField pointOnTheField) {
272
273             return new PointOnTheScreen(pointOnTheField.getX() * sizeOfBlock.getWidth(),
274                     pointOnTheField.getY() * sizeOfBlock.getHeight());
275         }
276
277         @Override
278         public void gameIsOver() {
279
280             isGameOver = true;
281         }
282
283         @Override
284         public void keyIsFound(PointOnTheField positionOfKey) {
285
286             isKeyFound = true;
287         }
288
289         @Override
290         public void doorIsOpen(PointOnTheField doorPosition) {
291
292             doorsClosed.remove(doorPosition);
293             doorsOpened.add(voidFieldNearPoint(doorPosition));
294
295             keys = model.getKeys();
296         }
297
298         /**
299          * @param point точка на игровом поле
300          * @return ближайшая клетка, на которую не может сходить протигонист
301          */
302         private PointOnTheField voidFieldNearPoint(PointOnTheField point) {
303
304             int x = point.getX();
305             int y = point.getY();
306
307             if(!model.isItPassableCells(x + 1, y))  return new PointOnTheField(x + 1, y);
```

```
308             if(!model.isItPassableCells(x, y + 1))  return new PointOnTheField(x, y + 1);
309             if(!model.isItPassableCells(x - 1, y))  return new PointOnTheField(x - 1, y);
310             if(!model.isItPassableCells(x, y - 1))  return new PointOnTheField(x, y - 1);
311
312             return point;
313         }
314
315         /**
316          * @return число ключей, которые собрал игрок
317          */
318         public int getNumberOfKeys() {
319
320             return model.getNumberOfKeys();
321         }
322 }
```

```
1  package com.maltsev.labyrinth.presenter;
2
3
4  /**
5   * Класс, созданый для того, чтобы парсить файл с полем,
6   * получать из него игровое поле по номеру
7   */
8  public class ParsingFile {
9
10     public static String getFieldOnTheNumber(int numberOfTheField) {
11
12         String fileData = FileReader.read("gameField.txt");
13
14         String field = "";
15
16         return fileData;
17     }
18 }
```

```
1  package com.maltsev.labyrinth.presenter;
2
3  import com.badlogic.gdx.Gdx;
4  import com.badlogic.gdx.files.FileHandle;
5
6  /**
7   *  Обработка данных из файла
8   */
9  class FileReader {
10
11     /**
12      * Чтение из файла
13      * @param fileName имя файла
14      * @return полученная информация из файла, в виде одной строки
15      */
16     static String read(String fileName) {
17
18         FileHandle file = Gdx.files.internal(fileName);
19
20         return file.readString();
21     }
22 }
```

```
1  package com.maltsev.labyrinth.presenter.tempdata;
2
3
4  public class PointOnTheScreen {
5
6      private float x;
7      private float y;
8
9      public PointOnTheScreen(float x, float y) {
10
11         this.x = x;
12         this.y = y;
13     }
14
15     public PointOnTheScreen(SizeOfTexture obj) {
16
17         this.x = obj.getWidth();
```

```
18          this.y = obj.getHeight();
19      }
20
21      public float getX() {
22
23          return x;
24      }
25
26      public float getY() {
27
28          return y;
29      }
30  }
```

```
1  package com.maltsev.labyrinth.presenter.tempdata;
2
3
4  public class SizeOfTexture {
5
6      private int width;
7      private int height;
8
9      public SizeOfTexture(int width, int height) {
10
11         this.width = width;
12         this.height = height;
13     }
14
15     public SizeOfTexture(SizeOfTexture newObj) {
16
17         this.width = newObj.getWidth();
18         this.height = newObj.getHeight();
19     }
20
21     public int getWidth() {
22
23         return width;
24     }
25
26     public int getHeight() {
27
28         return height;
29     }
30 }
```

```
1  package com.maltsev.labyrinth.presenter.interfaces;
2
3
4  import com.maltsev.labyrinth.presenter.tempdata.PointOnTheScreen;
5  import com.maltsev.labyrinth.presenter.tempdata.SizeOfTexture;
6
7  /**
8   * Интерфейс для взаимодействия с Presenter
9   */
10 public interface View {
11
12     /**
13      * @return размер блока текстуры
14      */
15     SizeOfTexture getSizeOfBlock();
16
17     /**
18      * Отрисовка блока текстуры
19      * @param point точка, в которой следует отрисовать блок
20      */
21     void drawBlock(PointOnTheScreen point);
22
23     /**
24      * Отрисовка финишной клетки
25      * @param point точка, в которой следует отрисовать финишную клетку
26      */
27     void drawExit(PointOnTheScreen point);
28
29     /**
30      * Отрисовка ключа
```

```
31          * @param point точка, в которой следует отрисовать ключ
32          */
33         void drawKey(PointOnTheScreen point);
34 
35         /**
36          * Отрисовка закрытой двери
37          * @param point точка, в которой следует отрисовать дверь
38          */
39         void drawCloseDoor(PointOnTheScreen point);
40 
41         /**
42          * Отрисовка открытой двери
43          * @param point точка, в которой следует отрисовать дверь
44          */
45         void drawOpenDoor(PointOnTheScreen point);
46 
47         /**
48          * Запрет на ввод
49          */
50         void lockInput();
51 
52         /**
53          * Отключить запрет на ввод
54          */
55         void unlockInput();
56 
57         /**
58          * Начать движение
59          */
60         void startMovement();
61 
62         /**
63          * Закончить движение
64          */
65         void finishMovement();
66 
67         /**
68          * Сообщение об окончание игры
69          */
70         void messageOfGameOver();
71 }
```

```
1  package com.maltsev.labyrinth.view;
2 
3 
4  import com.badlogic.gdx.Game;
5  import com.badlogic.gdx.graphics.g2d.SpriteBatch;
6  import com.maltsev.labyrinth.view.screens.GameScreen;
7  import com.maltsev.labyrinth.view.screens.MainMenuScreen;
8 
9  /**
10  * Главный класс из пакета view
11  *
12  * Представление, как правило, реализуется в Activity,
13  * которая содержит ссылку на презентер.
14  * Единственное, что делает представление,
15  * это вызывает методы презентера при какомлибо– действии пользователя
16  */
17 public class Labyrinth extends Game {
18 
19     public static final int V_WIDTH = 1920;
20     public static final int V_HEIGHT = 1080;
21 
22     private MainMenuScreen mainMenuScreen;
23     private GameScreen gameScreen;
24 
25     public SpriteBatch spriteBatch;
26 
27     @Override
28     public void create() {
29 
30         spriteBatch = new SpriteBatch();
31 
32         setMainMenuScreen();
33     }
34 
```

```
35      public void setGameScreen(int numberOfgameField) {
36
37          gameScreen = new GameScreen(this, numberOfgameField);
38          this.setScreen(gameScreen);
39
40      }
41
42      public void setMainMenuScreen() {
43
44          mainMenuScreen = new MainMenuScreen(this);
45          this.setScreen(mainMenuScreen);
46      }
47
48      @Override
49      public void render() {
50
51          super.render();
52      }
53
54      @Override
55      public void dispose() {
56
57          spriteBatch.dispose();
58
59          super.dispose();
60      }
61  }
```

```
1   package com.maltsev.labyrinth.view.screens;
2
3
4   import com.badlogic.gdx.Gdx;
5   import com.badlogic.gdx.Screen;
6   import com.badlogic.gdx.graphics.OrthographicCamera;
7   import com.badlogic.gdx.graphics.Texture;
8   import com.badlogic.gdx.graphics.g2d.SpriteBatch;
9   import com.badlogic.gdx.graphics.g2d.TextureAtlas;
10  import com.badlogic.gdx.math.Vector3;
11  import com.badlogic.gdx.scenes.scene2d.InputEvent;
12  import com.badlogic.gdx.scenes.scene2d.Stage;
13  import com.badlogic.gdx.scenes.scene2d.ui.ImageButton;
14  import com.badlogic.gdx.scenes.scene2d.ui.Skin;
15  import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;
16  import com.maltsev.labyrinth.presenter.Presenter;
17  import com.maltsev.labyrinth.presenter.interfaces.View;
18  import com.maltsev.labyrinth.presenter.tempdata.PointOnTheScreen;
19  import com.maltsev.labyrinth.view.Labyrinth;
20  import com.maltsev.labyrinth.view.scenes.Fon;
21  import com.maltsev.labyrinth.view.scenes.Hud;
22  import com.maltsev.labyrinth.presenter.tempdata.SizeOfTexture;
23
24  /**
25   * Игровой экран
26   * Здесь происходит основное действо
27   */
28  public class GameScreen implements Screen, View {
29
30      private Labyrinth game;
31      private Presenter presenter;
32      private Hud hud;
33      private Fon fon;
34
35      private SpriteBatch batch;
36      private OrthographicCamera camera;
37
38      private Texture block;
39      private SizeOfTexture sizeOfBlock;
40      private Texture exit;
41      private Texture doorClose;
42      private Texture doorOpen;
43      private Texture key;
44      private Texture protagonist;
45      private Texture infoGameEnd;
46      private Vector3 touchPos;
47      private Vector3 positionOfProtagonist;
48
```

```
49      private boolean lockInput = false;
50      private boolean isGameEnd = false;
51
52      private boolean isInMotion = false;
53
54      public GameScreen(final Labyrinth game, int numberOfGameField) {
55
56          this.game = game;
57
58          batch = game.spriteBatch;
59
60          camera = new OrthographicCamera();
61          camera.setToOrtho(false, Labyrinth.V_WIDTH, Labyrinth.V_HEIGHT);
62
63          hud = new Hud(batch, this);
64          fon = new Fon(batch);
65
66          touchPos = new Vector3();
67
68          positionOfProtagonist = new Vector3();
69
70          block = new Texture("game_ui/block.png");
71          sizeOfBlock = new SizeOfTexture(block.getWidth(), block.getHeight());
72
73          exit = new Texture("game_ui/exit.png");
74          doorClose = new Texture("game_ui/doorClose.png");
75          doorOpen = new Texture("game_ui/doorOpen.png");
76          key = new Texture("game_ui/key.png");
77          protagonist = new Texture("game_ui/protagonist.png");
78          infoGameEnd = new Texture("game_ui/grey_panel.png");
79
80          presenter = new Presenter(this, 1);
81
82          changePositionOfProtagonist(presenter.getPositionOfProtagonist());
83
84          Gdx.input.setInputProcessor(hud.stage);
85
86          camera.position.set(positionOfProtagonist);
87          camera.update();
88      }
89
90      @Override
91      public SizeOfTexture getSizeOfBlock() {
92
93          return sizeOfBlock;
94      }
95
96      @Override
97      public void lockInput() {
98
99          lockInput = true;
100     }
101
102     @Override
103     public void unlockInput() {
104
105         lockInput = false;
106     }
107
108     @Override
109     public void startMovement() {
110
111         isInMotion = true;
112     }
113
114     @Override
115     public void finishMovement() {
116
117         isInMotion = false;
118     }
119
120     private void changePositionOfProtagonist(PointOnTheScreen pointOfNewLocationOfProtagonist)
        ↪ {
121
122         positionOfProtagonist.x = pointOfNewLocationOfProtagonist.getX();
123         positionOfProtagonist.y = pointOfNewLocationOfProtagonist.getY();
```

```
124     }
125
126     private void handelInput(float delta) {
127
128         if (Gdx.input.justTouched() && !lockInput) {
129
130             touchPos.set(Gdx.input.getX(), Gdx.input.getY(), 0);
131
132             camera.unproject(touchPos);
133
134             presenter.moveProtagonist(touchPos.x, touchPos.y);
135         }
136     }
137
138     private void update(float delta) {
139
140         handelInput(delta);
141
142         if (isInMotion)
143             changePositionOfProtagonist(presenter.getPositionOfMovingProtagonist(delta));
144
145         hud.setTime(delta);
146
147         hud.setKeys(presenter.getNumberOfKeys());
148     }
149
150     /**
151      * Здесь происходит вся отрисовка
152      */
153     private void draw(float delta) {
154
155         presenter.drawField();
156         batch.draw(protagonist, positionOfProtagonist.x, positionOfProtagonist.y);
157         update(delta);
158
159         if(isGameEnd) {
160
161             batch.draw(infoGameEnd, positionOfProtagonist.x - infoGameEnd.getWidth()/2,
162                     positionOfProtagonist.y - infoGameEnd.getHeight()/2);
163
164             waitingAction();
165         }
166
167         camera.position.set(positionOfProtagonist);
168         camera.update();
169     }
170
171     private void waitingAction() {
172
173         if (Gdx.input.justTouched()) {
174
175             this.close();
176         }
177     }
178
179     @Override
180     public void render (float delta) {
181
182         fon.stage.draw();
183
184         batch.setProjectionMatrix(camera.combined);
185
186         // Следует вызывать методы отрисовки этого экрана только в пределах begin~end
187         batch.begin();
188         draw(delta);
189         batch.end();
190
191         hud.stage.draw();
192     }
193
194     @Override
195     public void drawBlock(PointOnTheScreen point) {
196
197         batch.draw(block, point.getX(), point.getY());
198     }
199
```

```
200     @Override
201     public void drawExit(PointOnTheScreen point) {
202
203         batch.draw(exit, point.getX(), point.getY());
204     }
205
206     @Override
207     public void drawKey(PointOnTheScreen point) {
208
209         batch.draw(key, point.getX(), point.getY());
210     }
211
212     @Override
213     public void drawCloseDoor(PointOnTheScreen point) {
214
215         batch.draw(doorClose, point.getX(), point.getY());
216     }
217
218     @Override
219     public void drawOpenDoor(PointOnTheScreen point) {
220
221         batch.draw(doorOpen, point.getX(), point.getY());
222     }
223
224     public void close() {
225
226         dispose();
227         game.setMainMenuScreen();
228     }
229
230     @Override
231     public void messageOfGameOver() {
232
233         isGameEnd = true;
234     }
235
236     @Override
237     public void dispose () {
238
239         hud.dispose();
240         fon.dispose();
241         block.dispose();
242         exit.dispose();
243         protagonist.dispose();
244         infoGameEnd.dispose();
245     }
246
247     @Override
248     public void show() {
249
250     }
251
252     @Override
253     public void resize(int width, int height) {
254
255     }
256
257     @Override
258     public void pause() {
259
260     }
261
262     @Override
263     public void resume() {
264
265     }
266
267     @Override
268     public void hide() {
269
270     }
271
272 }
```

```
1 package com.maltsev.labyrinth.view.screens;
2
```

```
3
4  import com.badlogic.gdx.Gdx;
5  import com.badlogic.gdx.Screen;
6  import com.badlogic.gdx.graphics.Color;
7  import com.badlogic.gdx.graphics.GL20;
8  import com.badlogic.gdx.graphics.Texture;
9  import com.badlogic.gdx.graphics.g2d.Batch;
10 import com.badlogic.gdx.graphics.g2d.BitmapFont;
11 import com.badlogic.gdx.graphics.g2d.TextureAtlas;
12 import com.badlogic.gdx.graphics.g2d.freetype.*;
13 import com.badlogic.gdx.scenes.scene2d.InputEvent;
14 import com.badlogic.gdx.scenes.scene2d.Stage;
15 import com.badlogic.gdx.scenes.scene2d.ui.*;
16 import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;
17 import com.badlogic.gdx.scenes.scene2d.utils.Drawable;
18 import com.badlogic.gdx.utils.viewport.StretchViewport;
19 import com.maltsev.labyrinth.view.Labyrinth;
20
21 /**
22  * Главное меню игры
23  */
24 public class MainMenuScreen implements Screen{
25
26     private Skin skinForButton;
27     private TextureAtlas atlasUiForButton;
28     private ImageTextButton.ImageTextButtonStyle buttonStyle;
29     private Stage stage;
30     private ImageTextButton play;
31     private Labyrinth game;
32     private BitmapFont font;
33     private Texture fon;
34     private Table table;
35     private ScrollPane scrollPane;
36     private List list;
37     private int numberOfgameField;
38
39
40     public MainMenuScreen(final Labyrinth game) {
41
42         this.game = game;
43
44         numberOfgameField = 1;
45
46         stage = new Stage(new StretchViewport(Labyrinth.V_WIDTH, Labyrinth.V_HEIGHT));
47
48         table = new Table();
49
50         fon = new Texture("menu_ui/fon.jpg");
51
52         atlasUiForButton = new TextureAtlas("menu_ui/menu.pack");
53
54         skinForButton = new Skin(atlasUiForButton);
55
56         buttonStyle = new ImageTextButton.ImageTextButtonStyle();
57
58         buttonStyle.up = skinForButton.getDrawable("blue_button04");
59         buttonStyle.down = skinForButton.getDrawable("blue_button04_down");
60
61         FreeTypeFontGenerator generator = new FreeTypeFontGenerator(Gdx.files.internal("font/
    ↪ some_font.ttf"));
62         FreeTypeFontGenerator.FreeTypeFontParameter parameter = new FreeTypeFontGenerator.
    ↪ FreeTypeFontParameter();
63         parameter.size = 65;
64         font = generator.generateFont(parameter);
65         generator.dispose();
66
67         buttonStyle.font = font;
68
69         play = new ImageTextButton("Start", buttonStyle);
70         play.addListener(new ClickListener() {
71
72             @Override
73             public void touchUp(InputEvent event, float x, float y, int pointer, int button) {
74
75                 game.setGameScreen(numberOfgameField);
76                 dispose();
```

```
77              }
78          });
79
80          table.center();
81          table.setFillParent(true);
82
83          Drawable drawable = new Drawable() {
84              @Override
85              public void draw(Batch batch, float x, float y, float width, float height) {
86
87              }
88
89              @Override
90              public float getLeftWidth() {
91                  return 0;
92              }
93
94              @Override
95              public void setLeftWidth(float leftWidth) {
96
97              }
98
99              @Override
100             public float getRightWidth() {
101                 return 0;
102             }
103
104             @Override
105             public void setRightWidth(float rightWidth) {
106
107             }
108
109             @Override
110             public float getTopHeight() {
111                 return 0;
112             }
113
114             @Override
115             public void setTopHeight(float topHeight) {
116
117             }
118
119             @Override
120             public float getBottomHeight() {
121                 return 0;
122             }
123
124             @Override
125             public void setBottomHeight(float bottomHeight) {
126
127             }
128
129             @Override
130             public float getMinWidth() {
131                 return 0;
132             }
133
134             @Override
135             public void setMinWidth(float minWidth) {
136
137             }
138
139             @Override
140             public float getMinHeight() {
141                 return 0;
142             }
143
144             @Override
145             public void setMinHeight(float minHeight) {
146
147             }
148         };
149
150         List.ListStyle listStyle = new List.ListStyle(font,new Color(1,1,1,1),
151                 new Color(0,0,0,1),drawable);
152         list = new List(listStyle);
```

```
153
154
155         String[] inputData = {"one","two","three"};
156
157         list.setItems(inputData);
158
159         ScrollPane.ScrollPaneStyle scrollPaneStyle = new ScrollPane.ScrollPaneStyle(drawable,
    ↪ drawable,drawable,drawable,drawable);
160         scrollPane= new ScrollPane(list, scrollPaneStyle);
161
162
163         //table.add(scrollPane);
164         table.add(play);
165
166         stage.addActor(table);
167
168         Gdx.input.setInputProcessor(stage);
169         Gdx.input.setCatchBackKey(true);
170     }
171
172     @Override
173     public void render(float delta) {
174
175         Gdx.gl.glClearColor(0.4f, 0.439f, 1f, 1);
176         Gdx.gl.glClear(GL20.GL_COLOR_BUFFER_BIT);
177
178         game.spriteBatch.begin();
179         game.spriteBatch.draw(fon, 0, 0);
180         game.spriteBatch.end();
181
182         stage.act(delta);
183         stage.draw();
184
185         //System.out.println(list.getSelected());
186     }
187
188     @Override
189     public void dispose() {
190
191         stage.dispose();
192         atlasUiForButton.dispose();
193         skinForButton.dispose();
194         font.dispose();
195     }
196
197     @Override
198     public void show() {
199
200
201     }
202
203     @Override
204     public void resize(int width, int height) {
205
206     }
207
208     @Override
209     public void pause() {
210
211     }
212
213     @Override
214     public void resume() {
215
216     }
217
218     @Override
219     public void hide() {
220
221     }
222 }
```

```
1 package com.maltsev.labyrinth.view.scenes;
2
3
4 import com.badlogic.gdx.graphics.OrthographicCamera;
```

```
5  import com.badlogic.gdx.graphics.Texture;
6  import com.badlogic.gdx.graphics.g2d.SpriteBatch;
7  import com.badlogic.gdx.scenes.scene2d.Stage;
8  import com.badlogic.gdx.scenes.scene2d.ui.Image;
9  import com.badlogic.gdx.utils.viewport.ScreenViewport;
10 import com.badlogic.gdx.utils.viewport.Viewport;
11
12 /**
13  * Класс, который отвечает за отрисовку фона
14  */
15 public class Fon {
16
17     public Stage stage;
18     private Viewport viewport;
19
20     private Image img;
21     private Texture fon;
22
23     public Fon(SpriteBatch spriteBatch) {
24
25         viewport = new ScreenViewport(new OrthographicCamera());
26         stage = new Stage(viewport, spriteBatch);
27
28         fon = new Texture("fon/grass.png");
29         img = new Image(fon);
30
31         img.setPosition(0,0);
32
33         stage.addActor(img);
34     }
35
36     /**
37      * Использовать при окончание работы с объектом
38      */
39     public void dispose() {
40
41         fon.dispose();
42         stage.dispose();
43     }
44 }
```

```
1  package com.maltsev.labyrinth.view.scenes;
2
3
4  import com.badlogic.gdx.Gdx;
5  import com.badlogic.gdx.graphics.Color;
6  import com.badlogic.gdx.graphics.OrthographicCamera;
7  import com.badlogic.gdx.graphics.Texture;
8  import com.badlogic.gdx.graphics.g2d.BitmapFont;
9  import com.badlogic.gdx.graphics.g2d.SpriteBatch;
10 import com.badlogic.gdx.graphics.g2d.TextureAtlas;
11 import com.badlogic.gdx.graphics.g2d.freetype.FreeTypeFontGenerator;
12 import com.badlogic.gdx.scenes.scene2d.InputEvent;
13 import com.badlogic.gdx.scenes.scene2d.Stage;
14 import com.badlogic.gdx.scenes.scene2d.ui.*;
15 import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;
16 import com.badlogic.gdx.utils.viewport.ScreenViewport;
17 import com.badlogic.gdx.utils.viewport.Viewport;
18 import com.maltsev.labyrinth.view.screens.GameScreen;
19
20 /**
21  * Класс, который отвечает за отрисовку обёрткивывод( информирующих лейлбов поверх игрового экрана)
22  */
23 public class Hud {
24
25     public Stage stage;
26     private BitmapFont font;
27
28     private double timeCount =0;
29
30     private Label keys;
31     private Label timer;
32
33     private Table tableTop;
34
35     private ImageButton pauseButton;
```

```
36      private TextureAtlas atlasUi;
37      private Skin skin;
38
39      private ClickListener pauseListener;
40
41      private Texture fon;
42
43      private GameScreen gameScreen;
44
45
46      public Hud(SpriteBatch spriteBatch, GameScreen gameScreen) {
47
48          this.gameScreen = gameScreen;
49
50          generateFont();
51
52          Viewport viewport = new ScreenViewport(new OrthographicCamera());
53          stage = new Stage(viewport, spriteBatch);
54
55          Label.LabelStyle labelStyle = new Label.LabelStyle(font, Color.WHITE);
56
57          timer  = new Label("",labelStyle);
58          keys = new Label("", labelStyle);
59
60          fon = new Texture("hud_gui/fon.png");
61          final Image img = new Image(fon);
62
63          atlasUi = new TextureAtlas("hud_gui/hud_ui.pack");
64          skin = new Skin(atlasUi);
65          pauseButton = new ImageButton(skin.getDrawable("yellow_button09"), skin.getDrawable("
        ↪ yellow_button090"));
66
67          registeredListenerAgain();
68
69          tableTop = new Table();
70          tableTop.top();
71          tableTop.setFillParent(true);
72
73          tableTop.add(timer).width(100).expandX().top();
74          tableTop.add(keys).expandX().top();
75          tableTop.add(pauseButton).expandX();
76
77
78          stage.addActor(tableTop);
79      }
80
81      private void registeredListenerAgain(){
82          pauseListener = new ClickListener() {
83
84              @Override
85              public void touchUp(InputEvent event, float x, float y, int pointer, int button) {
86
87                  gameScreen.close();
88                  registeredListenerAgain();
89              }
90          };
91          pauseButton.addListener(pauseListener);
92      }
93
94      public void setTime(float delta) {
95
96          timeCount += delta;
97          timer.setText("time:␣" + String.format("%.0f%n", timeCount));
98      }
99
100     public void setKeys(int numberOfKeys) {
101
102         keys.setText("keys:␣" + numberOfKeys);
103     }
104
105     private void generateFont() {
106
107         FreeTypeFontGenerator generator = new FreeTypeFontGenerator(Gdx.files.internal("font/
        ↪ abc.ttf"));
108         FreeTypeFontGenerator.FreeTypeFontParameter parameter = new FreeTypeFontGenerator.
        ↪ FreeTypeFontParameter();
```

```
109             parameter.size = 60;
110             font = generator.generateFont(parameter);
111             generator.dispose();
112         }
113 
114         /**
115          * Использовать при окончание работы с объектом
116          */
117         public void dispose() {
118 
119             skin.dispose();
120             atlasUi.dispose();
121             font.dispose();
122             stage.dispose();
123             fon.dispose();
124         }
125 }
```