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

Программирование

Отчет по курсовой работе
Игра "Коридор"

Работу выполнил:
Жуйков А.А.
Группа:
23501/4
Преподаватель:
Вылегжанина К.Д.

Санкт-Петербург
2016

Содержание

1 Настольная игра "Коридор"

1.1 Краткое описание игры

Суть игры Коридор заключается в том, чтобы довести свою фишку из одного конца доски в другой и не дать сделать этого противнику. В свой ход мы можем либо двигаться, дабы поскорее достичь финиша, либо строить стены, из которых впоследствии и вырастает тот самый коридор, в котором должен заблудиться наш соперник. Не стоит забывать также и о том, что в Коридоре поле у вас общее, поэтому любое препятствие противнику может в итоге обернуться против вас самих.

1.2 Правила игры

- Цель игры.
  Первому дойти до финишной линии противоположной стороны игрового поля.

- Правила игры для 2 игроков.
  Перед началом игры игроки помещают свои фишки на среднюю клетку первого ряда на своей стороне поля и получают 10 перегородок. Игроки бросают жребий и определяют, кто начнет игру.

- Ход игры.
  В свой ход игрок может:

  1) Переместить свою фишку. Игрок в свой ход может переместить фишку на одну клетку вперед, назад, влево или вправо. Фишка не может "перепрыгнуть" через перегородку.
  2) Поставить на поле одну перегородку. Перегородка ставится так, чтобы закрыть ровно две клетки. Ее можно поставить так, чтобы облегчить путь себе, либо препятствовать движению соперника. При этом всегда следует оставить сопернику выход к финишной линии.

  Если игрок использовал все перегородки, он продолжает играть только фишкой.

- Лицом к лицу.
  Если фишки обоих игроков находятся на соседних клетках и между ними нет перегородки, игрок, которому принадлежит ход, может своей фишкой "перепрыгнуть" через фишку соперника и таким образом переместиться на еще одну клетку вперед. Если же сразу за фишкой соперника стоит перегородка, игрок может переместить свою фишку вправо или влево от фишки соперника.

- Окончание игры.
  Побеждает тот, кто первым доходит до какой-либо из 9 клеток финишной линии на противоположной стороне игрового поля.

- Правила для четырех игроков.
  В начале игры каждый игрок помещает одну фишку на среднюю клетку ближайшего к нему ряда игрового поля. Каждый игрок получает по 5 перегородок. В этот вариант играют согласно правилам игры для 2 игроков. Однако, фишка игрока может "перепрыгнуть"не более чем на одну фишку. Ход передается по часовой стрелке.

1.3 Задание

Разработать приложение, позволяющее играть в игру ”Коридор” двум игрокам, а также одному игроку против искусственного интеллекта. Вместо четырех игроков, было выбрано другое дополнение игры.

1.4 Дополнение игры.

В игру вводится еще один тип игрока - the Fox. The Fox появляется в случайной свободной клетке поля в указанный ход (от начала игры) и преследует случайного игрока на поле. The Fox ходит один раз в заданное количество ходов и не может ставить перегородки. Этого игрока, в отличие от простых игроков, можно закрыть за барьерами. The Fox побеждает, когда ловит игрока, поместив свою фишку на чужую. Таким образом, задача игрока усложняется: необходимо не только достичь финишной линии первым, но и ”сбежать” от the Fox.

1.5 Диаграмма прецедентов использования

На рисунках 1 и 2 показаны диаграммы прецедентов использования.

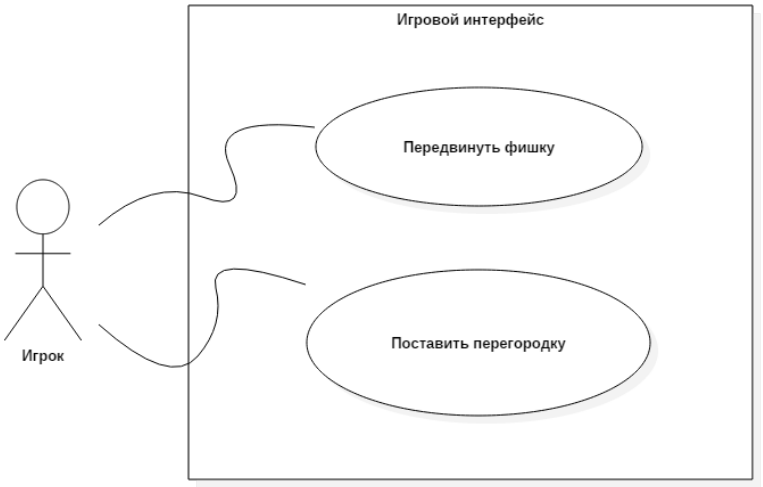

Рис. 1: Диаграмма прецедентов использования.

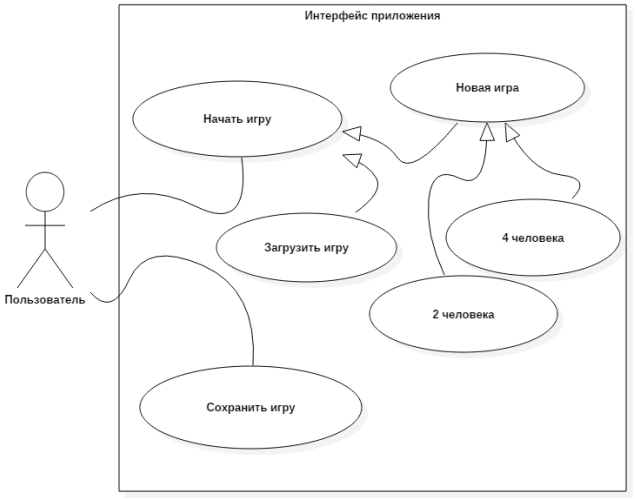

Рис. 2: Диаграмма прецедентов использования.

1.6 Вывод

Определено задание разработки приложения. Составлены диаграммы прецедентов использования.

2 Проектирование приложения, реализующего игру ”Коридор”

Приложение должно позволять играть в игру двум игрокам, а также против искусственного интеллекта.

2.1 Проектирование библиотеки

Библиотека - ядро приложения. Здесь содержатся основные классы, необходимые для представления игры.
В API выделены следующие методы:

- Метод, передвигающий фишку текущего игрока в заданные координаты поля. (Текущий игрок - игрок, чья очередь делать ход.)
- Метод, размещающий перегородку текущего игрока на клетку с заданными координатами и в заданном направлении (горизонтально или вертикально).
- Метод, возвращающий список доступных ходов фишкой текущего игрока.
- Метод, позволяющий получить информацию о заданном игроке.
- Метод, возвращающий текущего игрока.

- Метод, возвращающий *true*, если какой-либо игрок достиг своей финишной линии и $false$ в обратном случае.
- Методы, возвращающие информацию о the Fox: ход, на котором появляется the Fox, и частота, с которой the Fox делает ход.

2.2 Вывод

Былы выделены основные методы API.

3 Реализация игры "Коридор"

3.1 Среда разработки

Интегрированная среда разработки IntelliJ IDEA 2016.2.5.
Язык: Java 1.8.

3.2 Реализация консольного приложения

Консольное приложение предоставляет пользователю всю функциональность ядра и позволяет запускать игру в консоли.
Классы, необходимые для консольного приложения содержатся в пакете *ru.spbstu.icc.kspt.zhuikov.quoridor.console*. Основные классы, выделенные в консольном приложении:

- Класс ConsoleGame. Создает игру, организует консольное взаимодействие с пользователем.
- Класс ConsoleDrawer. Выводит в консоль состояние поля игры, а также другую информацию: количество перегородок у игроков, очередь хода, победителя.
- Классы Command и CommandReader. Классы, необходимые для взаимодействия пользователя с приложением посредством команд, вводимых в консоль. Класс Command представляет собой такую команду, состоящую из нескольких частей (например, имя команды, координаты), при этом некоторые из них могут отсутствовать. CommandReader проверяет введенную команду на правильность с помощью регулярных выражений. Примеры команд:

  marker 2 2

  передвигает фишку текущего игрока на позицию 2 2, если это возможно.

  BARRIER 5 5 HORIZONTAL

  ставит перегородку текущего игрока горизонтально на позицию 5 5, если это возможно.

Если команду невозможно выполнить (например, "перепрыгнуть" фишкой через перегородку), то в консоль выводится соответсвующее сообщение об ошибке.

На рисунках 3 и 4 представлено состояние поля до выполнения команды и после него.

```
TOP 10
0 0 0 0 T 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 B 0 0 0 0
BOTTOM 10
TOP player's turn: barrier 7 7 Horizontal
```

Рис. 3: Начальное состояние игры.

```
TOP 9
0 0 0 0 T 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0
      ###
0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0

0 0 0 0 B 0 0 0 0
BOTTOM 10
BOTTOM player's turn:
```

Рис. 4: Состояние игры после выполнения команды *barrier* 7 7 *Horizontal*.

3.3 Реализация библиотеки

Классы библиотеки объединены в пакет *ru.spbstu.icc.kspt.zhuikov.quoridor*. Основные классы, выделенные в библиотеке:

- Класс Quoridor. Реализует методы, заявленные в API. Содержит игровое поле, список всех игроков в порядке очереди хода. Также меняет текущего игрока после очередного хода.
- Класс QuoridorField. Класс представляет поле модели. Реализуется в виде двумерного массива клеток поля - private class Cell, в которых

содержится информация о находящемся на ней объекте и цвете. Класс также содержит размер поля. Присутствуют методы, ставящие на поле объект, занимающий одну клетку и объект, занимающий несколько клеток (см. рисунок 6); методы, возвращающие эти объекты, размер поля, цвет клетки поля.

- Перечисление QuoridorPlayer. Класс представления игрока в игре. Перечисление было выбрано потому, что игроков может быть всегда только ограниченное количество (2 или 4), их позиции всегда закреплены (TOP, BOTTOM...). У каждого игрока содержится следующая информация: начальные координаты фишки; номер финишной линии; список перегородок, их текущее количество; ссылка на поле; а также флаг, является ли игрок ботом.

  Игрок содержит следующие методы: методы, возвращающие фишку игрока, список барьеров; методы, позволяющие сделать ход. Использовалась перегрузка методов: в зависимости от количества переданных аргументов ставится перегородка или передвигается фишка. В классе игрок перед ходом проверяется возможность хода. Если ход сделать невозможно, генерируется соответствующее исключение с информацией об ошибке.

- Класс Fox. Класс для представления игрока the Fox в игре. Содержит фишку, фишку цели и ссылку на поле. Имеется метод, позволяющий сделать ход. Цель выбирается случайно. Для передвижения фишки, вычисляется кратчайший путь до цели и делается ход на необходимые координаты.

- Классы, представляющие объекты на поле. Эти классы содержатся в подпакете *.items* Иерархия классов показана на рисунке 6

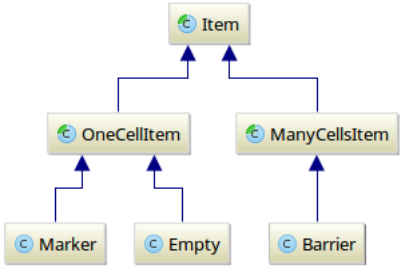

Рис. 5: Диаграмма классов, реализующих объекты на поле.

Класс OneCellItem содержит текущие координаты объекта, а также владельца (TOP, BOTTOM, NOBODY...). Класс ManyCellItem, в отличие от OneCellItem, содержит список координат объекта.

- Классы-исключения. Исключения, использующиеся в библиотеке приложения, содержатся в подпакете *.exceptions*.

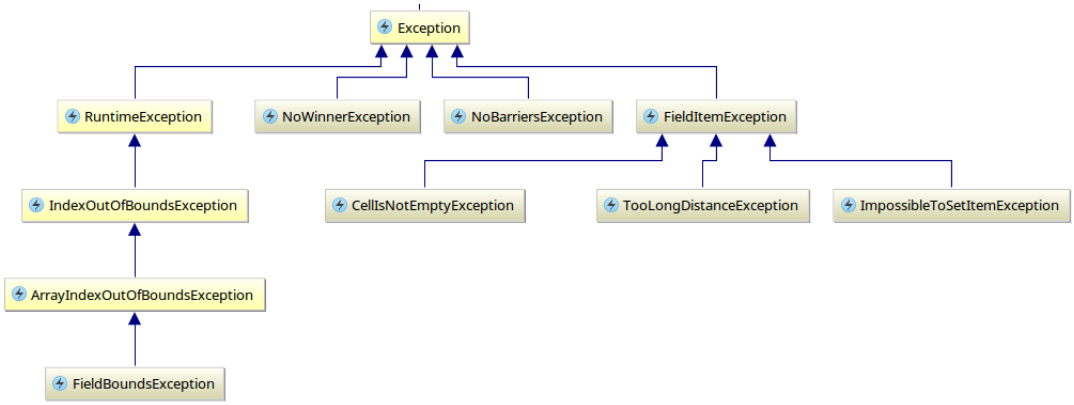

Рис. 6: Диаграмма классов, реализующих исключения.

3.4 Реализация графического приложения

Для создания графического приложения была выбрана библиотека Swing.

На рисунке 7 представлено главное окно приложения. Пользователю, предоставляется возможность начать одиночную игру против бота, начать игру для двух человек или выйти из игры.

Рис. 7: Главное меню графического приложения

На рисунке 8 - окно игры. Отображается информация о поле, количестве

барьеров, очередь хода. В строке под полем выводится различная информация, например, текущий игрок, информация о причине невозможности хода, победитель. Присутвует кнопка выхода в меню. Для того, чтобы поставить перегородку, необходимо выбрать его направление, нажать на кнопку ”Place barrier” и кликнуть на нужное место на поле.

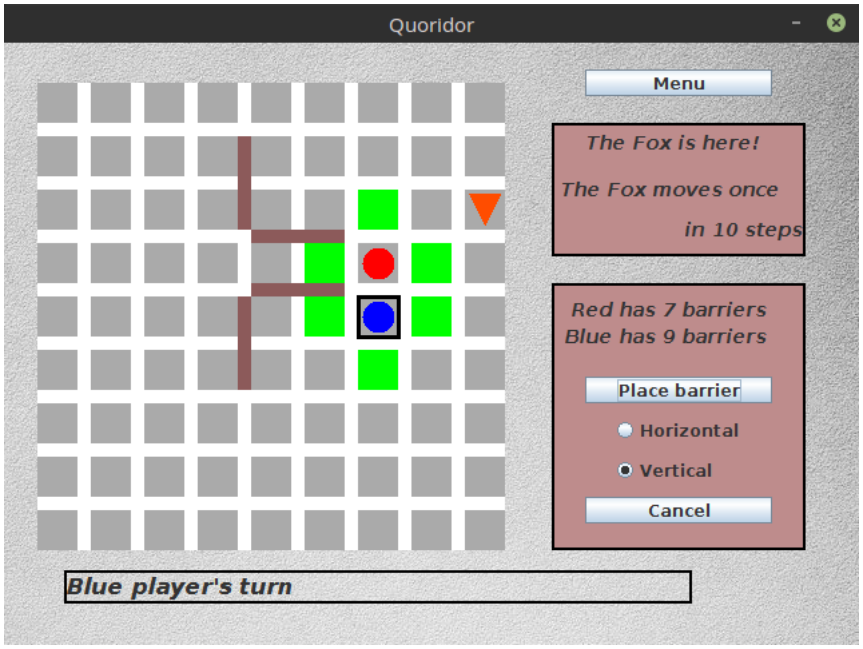

Рис. 8: Скриншот окна игры. Зеленым цветом подсвечиваются ходы, на которые можно сходить фишкой текущему игроку.

На рисунке 9 показана победа игрока the Fox.

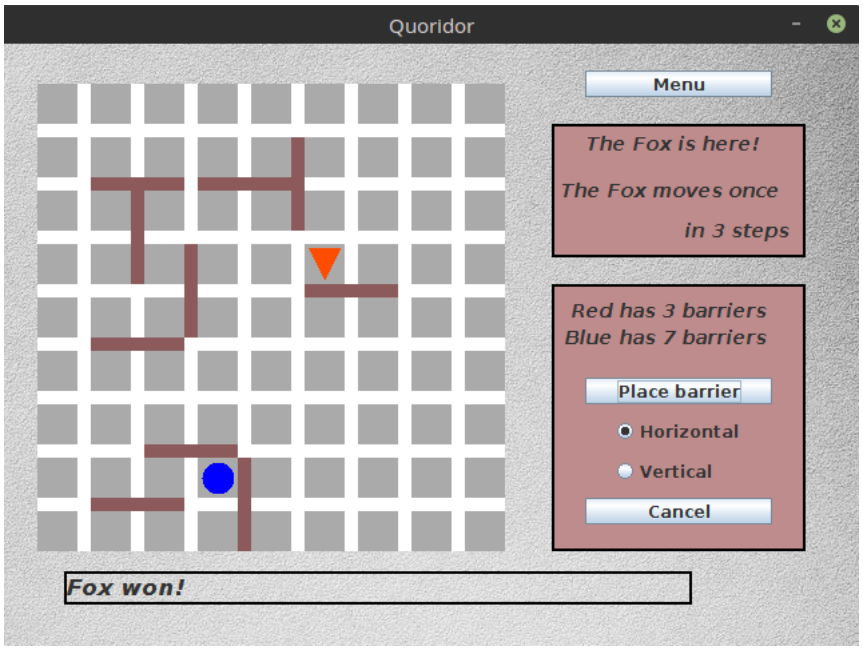

Рис. 9: Победа игрока the Fox: удалось догнать красного игрока.

На рисунке 11 показан скриншот игровой ситуации, в которой победу одержал красный игрок.

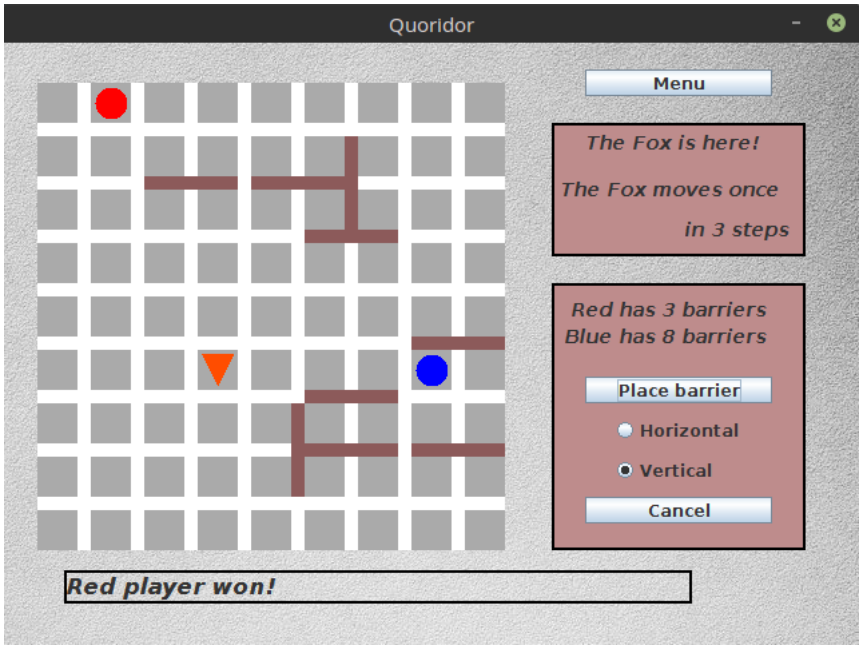

Рис. 10: Красный игрок одержал победу, достигнув финишной линии.

3.5 Вывод

Для реализации игры определены основные классы библиотеки, консольного и графического приложений. Разделение на подпакеты упрощает процесс работы над проектом, а также его структуру. Разработано графическое приложение, предоставляющее возможность играть одному игроку или двум игрокам.

4 Процесс обеспечения качества и тестирование

4.1 Просмотр кода

В ходе проектирования была проведена проверка исходного кода программы с целью обнаружения и исправления ошибок (code review). В результате было получено около 40-ти замечаний.

Большая часть замечаний исправлена.

4.2 Проведенные демонстрации

Была проведена одна презентация приложения. Получены следующие замечания: Последняя демонтрация перед релизом консольного приложения

Замечания:

- Выводить информацию о количестве барьеров обоих игроков, а не только текущего.
- Перегородки переносить курсором мыши на необходимое место.
- Подсвечивать возможные ходы выбранной фишки игрока.
- Добавить какой-нибудь искусственный интеллект и возможность играть одному игроку.
- Добавить еще одного игрока - the Fox.

Все недочеты и замечания, кроме переноски перегородки мышью, исправлены.

4.3 Тестирование кода приложения

Приложение содержит модульные тесты. Для тестирования был использован пакет org.junit.Test. Протестированы некоторые основные классы. Имеется большое количество тестов классов QuoridorField и QouridorPlayer: проверяется установка перегородки на поле, возможность перемещения фишки на позиции. С помощью возможнстей среды разработки было подсчитано покрытие кода тестами. Результаты представлены на рисунке **??**.

Element	Class, %	Method, %	Line, %
console	0% (0/7)	0% (0/19)	0% (0/103)
exceptions	85% (6/7)	85% (6/7)	85% (12/14)
gui	0% (0/20)	0% (0/102)	0% (0/415)
items	100% (9/9)	89% (17/19)	95% (46/48)
returningClasses	0% (0/3)	0% (0/13)	0% (0/32)
CellColor	100% (1/1)	100% (2/2)	100% (3/3)
Coordinates	100% (1/1)	83% (5/6)	83% (15/18)
Fox	0% (0/1)	0% (0/5)	0% (0/29)
Quoridor	0% (0/2)	0% (0/17)	0% (0/71)
QuoridorField	75% (3/4)	57% (11/19)	65% (66/101)
QuoridorPlayer	100% (1/1)	60% (14/23)	62% (98/158)

Рис. 11: Процент покрытия кода тестами.

4.4 Вывод

Тестирование приложения - важная часть разработки. Просмотр кода помогает найти какие-то недочеты в проектировании кода или в других вопросах. С помощью тестов можно быстро определить ошибку при изменении какой-либо части кода.

5 Вывод

Курсовая работа этого семестра – это отличная возможность ближе познакомиться с новым для автора языком Java. Как оказалось, язык намного понятнее и проще, чем уже известный C++.

Во время работы над проектом, автор приобрел практический опыт в области проектирования приложений, а также создания графических приложений на языке Java.

6 Приложение 1. Листинги кода

6.1 Библиотека

```
1  package ru.spbstu.icc.kspt.zhuikov.quoridor;
2
3  import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
       ↪ FieldItemException;
4  import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
       ↪ NoBarriersException;
5  import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
       ↪ NoWinnerException;
6  import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;
7  import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Field;
8  import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Player;
9  import java.util.ArrayList;
10 import java.util.List;
11
12
13 //TODO мне не хватает документации к коду
```

```
14 //TODO также хотелось бы, чтобы ядро с логикой было выделенно, если не в
       ↪ отдельный модуль, то хотябы в отдельный пакетв( отдельный
       ↪ относительно UI)
15
16 public class Quoridor {
17
18     private QuoridorField field = new QuoridorField(9);
19     private List<QuoridorPlayer> players = new ArrayList<
       ↪ QuoridorPlayer>(); //todo мб убрать этот ненужный список
20     private Fox fox;
21
22     //TODO возможно есть смысл поменять на enum, с методом nextPlayer()
       ↪ ;
23     private int currentPlayer;
24     private int step = 0;
25     private static int foxTime = 20;
26     private static int foxFrequency = 10;
27
28     public Quoridor(int playersNumber, boolean bot) {
29
30         if (playersNumber == 2) {
31             QuoridorPlayer player = QuoridorPlayer.TOP;
32             player.createPlayer(field, false);
33             currentPlayer = 0;                          //TODO несколько
       ↪ не очевидная строчка, может логичнее  = 1;
34             players.add(player);
35
36             player = QuoridorPlayer.BOTTOM;
37             player.createPlayer(field, bot);
38             players.add(player);
39
40         } else {
41             throw new UnsupportedOperationException("пока␣рано␣еще␣
       ↪ думать␣о␣чемто-␣большем...");
42         }
43     }
44
45     public Player getCurrentPlayer() {
46
47         switch (players.get(currentPlayer)) {
48             case TOP:
49                 return Player.TOP;
50             case BOTTOM:
51                 return Player.BOTTOM;
52 //              case RIGHT:
53 //                  return Player.RIGHT;
54 //              default:
55 //                  return Player.LEFT;
56         }
57         throw new AssertionError("unknown␣player" + players.get(
       ↪ currentPlayer));
58     }
59
60     public static void setFoxTime(int foxTime) {
61         Quoridor.foxTime = foxTime;
62     }
63
64     public static int getFoxTime() {
65         return foxTime;
66     }
67
68     public static int getFoxFrequency() {
```

```
69          return foxFrequency;
70      }
71
72      public static void setFoxFrequency(int foxTurn) {
73          Quoridor.foxFrequency = foxTurn;
74      }
75
76      public int getStep() {
77          return step;
78      }
79
80      public Field getField() {
81          return new Field(field);
82      }
83
84      //todo по рукам бы надавать наверно надо за такое...
85      public Player getPlayerInformation(Player player) {
86
87          switch (player) {
88              case TOP:
89                  player.createPlayer(QuoridorPlayer.TOP.
    ↪ getBarriersNumber());
90                  return player;
91              case BOTTOM:
92                  player.createPlayer(QuoridorPlayer.BOTTOM.
    ↪ getBarriersNumber());
93                  return player;
94
95  //              case RIGHT:
96  //                  player.createPlayer(QuoridorPlayer.RIGHT.
    ↪ getBarriersNumber());
97  //                  return player;
98  //              default:
99  //                  player.createPlayer(QuoridorPlayer.LEFT.
    ↪ getBarriersNumber());
100 //                  return player;
101         }
102         throw new AssertionError("unknown player" + player);
103     }
104
105     public boolean isEnd() {                //TODO возможно следует
    ↪ подумать об использование шаблона Наблюдатель
106
107         if (QuoridorPlayer.TOP.getMarker().getCoordinates().
    ↪ getVertical() == QuoridorPlayer.TOP.getDestinationRow()) {
108             return true;
109         }
110
111         if (QuoridorPlayer.BOTTOM.getMarker().getCoordinates().
    ↪ getVertical() == QuoridorPlayer.BOTTOM.getDestinationRow())
    ↪ {
112             return true;
113         }
114
115         if (fox != null && fox.getMarker().getCoordinates().equals(
    ↪ fox.getTarget())) {
116             return true;
117         }
118
119         // todo тут еще потом других сделать
120         return false;
121     }
```

```
122
123     public Player getWinner() throws NoWinnerException {    //TODO
        ↪ помоему-, при использование Наблюдателя метод атрофируется
124
125         if (isEnd()) {
126             if (QuoridorPlayer.TOP.getMarker().getCoordinates().
        ↪ getVertical() == field.getRealSize() - 1) {
127                 return Player.TOP;
128             }
129             if (QuoridorPlayer.BOTTOM.getMarker().getCoordinates().
        ↪ getVertical() == 0) {
130                 return Player.BOTTOM;
131             }
132             if (fox != null && fox.getMarker().getCoordinates().
        ↪ equals(fox.getTarget())) {
133                 return Player.FOX;
134             }
135         }
136
137         // todo вообще ужас! за это точно надо надавать...
138         throw new NoWinnerException("There is no winner");
139     }
140
141     public void moveMarker(int vertical, int horizontal)
142             throws FieldItemException, NoBarriersException {  //
        ↪ TODO странное название у исключения, возможно есть смысл
        ↪ переименовать в Выход за границу поля, но это не точно
143
144         players.get(currentPlayer).makeMove(vertical, horizontal);
145         changePlayerTurn();
146     }
147
148     public void placeBarrier(int vertical, int horizontal,
        ↪ BarrierPosition position)
149             throws FieldItemException, NoBarriersException {
150
151         players.get(currentPlayer).makeMove(vertical, horizontal,
        ↪ position);
152         changePlayerTurn();
153     }
154
155     public List<Coordinates> getPossibleMoves() {
156
157         return players.get(currentPlayer).getPossibleMoves();
158     }
159
160     private void changePlayerTurn() throws FieldItemException,
        ↪ NoBarriersException {
161
162         if (++currentPlayer == players.size()) {
163             currentPlayer = 0;
164         }
165
166         if (isEnd()) return;
167
168         if (fox != null && step % foxFrequency == 0) {
169             fox.makeMove();
170         }
171
172         if (step == foxTime) {
173             fox = new Fox(field);
174         }
```

```
175
176             step++;
177
178             if (players.get(currentPlayer).isBot()) {
179                 players.get(currentPlayer).makeBotMove();
180                 changePlayerTurn();
181             }
182         }
183
184 }
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor;


import ru.spbstu.icc.kspt.zhuikov.quoridor.items.*;

import java.util.*;

public class QuoridorField {
    //TODO я бы засунул этот класс в пакет связаный с полем да( и вообще всё, что свзяано с полем засунул в один пакет)

    private final int realSize;
    private final int size;
    private Cell[][] field;     //TODO почему бы не использовать, какуюнибудь– коллекцию? конечно( банально, но например, List<List<~>>)

    public QuoridorField(int size) {

        this.size = size;
        realSize = size * 2 - 1;
        field = new Cell[realSize][realSize];
        for (int i = 0; i <= realSize - 1; i++) {
            for (int j = 0; j <= realSize - 1; j++) {
                if ((i % 2 == 0) && (j % 2 == 0)) {
                    field[i][j] = new Cell(CellColor.BLACK, new Empty(i, j));
                } else {
                    field[i][j] = new Cell(CellColor.WHITE, new Empty(i, j));
                }
            }
        }
    }

    public QuoridorField() {
        this(9);
    }

    private class Cell {
        private CellColor color;
        private Item item;

        Cell(CellColor color, Item item) {
            this.color = color;
            this.item = item;
        }
    }

    public void setItem(OneCellItem item) {
        field[item.getCoordinates().getVertical()][item.getCoordinates().getHorizontal()].item = item;
    }

    public void setItem(ManyCellsItem item) {
        for (Coordinates coordinates : item.getCoordinates()) {
            field[coordinates.getVertical()][coordinates.getHorizontal()].item = item;
        }
    }

    public void clearCell(int vertical, int horizontal) {
```

```
56          field[vertical][horizontal].item = new Empty(vertical,
    ↪ horizontal);
57      }
58
59      public void clearCells(List<Coordinates> coordinatesList) {
60          for (Coordinates coordinates : coordinatesList) {
61              field[coordinates.getVertical()][coordinates.
    ↪ getHorizontal()].item
62                      = new Empty(coordinates.getVertical(),
    ↪ coordinates.getHorizontal());
63          }
64      }
65
66      public Item getItem(int vertical, int horizontal) {
67
68          //TODO к примеру, здесь был бы get(), он, по идее, безопаснее
69
70          return field[vertical][horizontal].item;  // todo мб return
    ↪  new ...
71      }
72
73      public CellColor getColor(int vertical, int horizontal) {
74          return field[vertical][horizontal].color;
75      }
76
77      public int getRealSize() {  //TODO не вижу смысла, в слове Real,
    ↪ юезр поля не должен знать, что тут творится у( тебя же нет другого
    ↪ метода получения размера поля), мне бы больше понравился простой
    ↪ getSize()
78          return realSize;
79      }
80
81      public int getSize() {
82          return size;
83      }
84
85      /**
86       * Возвращает кратчайший путь в( координатах) из заданной позиции до
    ↪ заданного ряда.
87       * Если нельзя пройти, вернется пустой стек.
88       * @param marker — координаты, из которых ищется путь
89       * @param rowNumber — номер ряда строки()
90       */
91      public Stack<Coordinates> getPathToRow(Coordinates marker, int
    ↪ rowNumber) {
92
93          if (marker.getVertical() == rowNumber) {
94              Stack<Coordinates> coordinates = new Stack<>();
95              coordinates.add(getNeighbours(marker).get(0));
96              return coordinates;
97          }
98
99          class Vertex {
100             public Coordinates coordinates;
101             public Vertex from;
102
103             public Vertex(Coordinates coordinates, Vertex from) {
104                 this.coordinates = coordinates;
105                 this.from = from;
106             }
107         }
108
```

```
109         boolean used[][] = new boolean[realSize][realSize];
110         Queue<Vertex> queue = new LinkedList<>();
111         List<Vertex> usedVertexes = new ArrayList<>();
112         Stack<Coordinates> path = new Stack<>();
113 
114         queue.add(new Vertex(marker, null));
115 
116         while (!queue.isEmpty()) {
117 
118 //          for (Coordinates coordinates : queue) {
119 //              System.out.print(coordinates + " ");
120 //          }
121 //          System.out.println();
122 
123             if (queue.element().coordinates.getVertical() ==
    ↪ rowNumber) {
124                 Vertex vertex = queue.element();
125                 while (vertex.from != null) {
126                     path.add(vertex.coordinates);
127                     vertex = vertex.from;
128                 }
129                 return path;
130             }
131 
132             for (final Coordinates neighbour : getNeighbours(queue.
    ↪ element().coordinates)) {
133                 try {
134                     if (!used[neighbour.getVertical()][neighbour.
    ↪ getHorizontal()] &&   // todo: шлифануть бы тут
135                             getItem((queue.element().coordinates.
    ↪ getVertical() + neighbour.getVertical()) / 2,
136                                     (queue.element().coordinates.
    ↪ getHorizontal() + neighbour.getHorizontal()) / 2).getType()
    ↪ != ItemType.BARRIER &&
137                             !queue.contains(new Vertex(neighbour,
    ↪ queue.element()))) {
138                                     //Coordinates(neighbour.
    ↪ getVertical(), neighbour.getHorizontal()))) {
139                         queue.add(new Vertex(neighbour, queue.
    ↪ element())); //neighbour
140                     }
141                 } catch (ArrayIndexOutOfBoundsException e) { }
142             }
143 
144             used[queue.element().coordinates.getVertical()][queue.
    ↪ element().coordinates.getHorizontal()] = true;
145             usedVertexes.add(queue.element());
146             queue.remove();
147         }
148 
149         return path;
150     }
151 
152     /**
153      * Возвращает кратчайший путь.
154      * @param init начальные координаты
155      * @param dest конечные координаты
156      * @return стек координат, по которым нужно пройти.
157      * Если пройти нельзя, вернется пустой стек.
158      */
159 
160     //это метод для лисы, надо будет сделать, чтобы все его использовали
```

```
161     //для этого выбросить тот енам наверно
162     public Stack<Coordinates> getPath(Coordinates init, Coordinates
        ↪  dest) {
163
164         class Vertex {
165             private Coordinates coordinates;
166             private Vertex from;
167
168             private Vertex(Coordinates coordinates, Vertex from) {
169                 this.coordinates = coordinates;
170                 this.from = from;
171             }
172         }
173
174         boolean used[][] = new boolean[realSize][realSize];
175         Queue<Vertex> queue = new LinkedList<>();
176         List<Vertex> usedVertexes = new ArrayList<>();
177         Stack<Coordinates> path = new Stack<>();
178
179         queue.add(new Vertex(init, null));
180
181         while (!queue.isEmpty()) {
182
183             if (queue.element().coordinates.equals(dest)) {
184                 Vertex vertex = queue.element();
185                 while (vertex.from != null) {
186                     path.add(vertex.coordinates);
187                     vertex = vertex.from;
188                 }
189                 return path;
190             }
191
192             for (final Coordinates neighbour : getNeighbours(queue.
        ↪ element().coordinates)) {
193                 try {
194                     if (!used[neighbour.getVertical()][neighbour.
        ↪ getHorizontal()] &&
195                             getItem((queue.element().coordinates.
        ↪ getVertical() + neighbour.getVertical()) / 2,
196                                     (queue.element().coordinates.
        ↪ getHorizontal() + neighbour.getHorizontal()) / 2).getType()
        ↪ != ItemType.BARRIER &&
197                             !queue.contains(new Vertex(neighbour,
        ↪ queue.element()))) {
198                         //Coordinates(neighbour.getVertical(),
        ↪ neighbour.getHorizontal()))) {
199                         queue.add(new Vertex(neighbour, queue.
        ↪ element())); //neighbour
200                     }
201                 } catch (ArrayIndexOutOfBoundsException e) { }
202             }
203
204             used[queue.element().coordinates.getVertical()][queue.
        ↪ element().coordinates.getHorizontal()] = true;
205             usedVertexes.add(queue.element());
206             queue.remove();
207         }
208
209         return path;
210     }
211
212     private List<Coordinates> getNeighbours(Coordinates coordinates
```

```
    ↪ ) {
213
214         List<Coordinates> neighbours = new ArrayList<Coordinates>()
    ↪ ;
215
216         neighbours.add(new Coordinates(coordinates.getVertical() -
    ↪ 2, coordinates.getHorizontal()));
217         neighbours.add(new Coordinates(coordinates.getVertical(),
    ↪ coordinates.getHorizontal() - 2));
218         neighbours.add(new Coordinates(coordinates.getVertical() +
    ↪ 2, coordinates.getHorizontal()));
219         neighbours.add(new Coordinates(coordinates.getVertical(),
    ↪ coordinates.getHorizontal() + 2));
220
221         return neighbours;
222     }
223
224 }
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor;


import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.*;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.*;

import java.util.*;


// todo разбить бы весь этот огромный класс. Мб сделать отдельный класс с логикой
// todo вообще, надо было без енама
public enum QuoridorPlayer {

    TOP(0, 8, Owner.TOP, 16),
    BOTTOM(16, 8, Owner.BOTTOM, 0);
  //  RIGHT(8, 16, Owner.RIGHT),
  //  LEFT(8, 0, Owner.LEFT);

    private final int initialVertical;
    private final int initialHorizontal;
    private final Owner owner;
    private final int destinationRow;

    private Marker marker;
    private boolean isBot = false;
    private int barriersNumber = 10;
    private List<Barrier> barriers = new ArrayList<Barrier>();
    private QuoridorField field;
    private boolean isActive = false;

    QuoridorPlayer(int initialVertical, int initialHorizontal, Owner owner, int destinationRow) {
        this.initialVertical = initialVertical;
        this.initialHorizontal = initialHorizontal;
        this.owner = owner;
        this.destinationRow = destinationRow;
    }

    public void createPlayer(QuoridorField field, boolean isBot) {
                //TODO смущает, что чтобы создать игрока ему нужно передать поле
        this.field = field;
        this.isBot = isBot;
        this.isActive = true;
        barriersNumber = 10;
        marker = new Marker(initialVertical, initialHorizontal, owner);
        field.setItem(marker);
    }

    public Marker getMarker() { return marker; }

    public int getBarriersNumber() { return barriersNumber; }

    public int getDestinationRow() {
        return destinationRow;
    }

    public List<Barrier> getBarriers() { return barriers; }

    public boolean isActive() { return isActive; }
```

```
58
59      public boolean isBot() { return  isBot; }
60
61      public List<Coordinates> getPossibleMoves() {
62
63          List<Coordinates> possibleMoves = new LinkedList<>();
64          for (int i = this.marker.getCoordinates().getVertical() -
    ↪ 4; i <= this.marker.getCoordinates().getVertical() + 4; i++)
    ↪  {
65              for (int j = this.marker.getCoordinates().getHorizontal
    ↪ () - 4; j <= this.marker.getCoordinates().getHorizontal() +
    ↪ 4; j++) {
66                  try {
67                      checkPlace(i, j);
68                      possibleMoves.add(new Coordinates(i, j));
69                  } catch (Exception e) {}
70              }
71          }
72
73          return possibleMoves;
74      }
75
76      public void makeBotMove() throws FieldItemException,
    ↪ NoBarriersException {
77
78          if (!isBot) {
79              throw new UnsupportedOperationException(this.name() + "
    ↪ ␣not␣a␣bot");
80          }
81          double rand = Math.random();
82          if (rand > 0.45) {
83              setBotBarrier();
84          } else setBotMarker();
85      }
86
87      private void setBotMarker() throws FieldItemException {
88
89          Stack<Coordinates> path = field.getPathToRow(marker.
    ↪ getCoordinates(), destinationRow);
90          if (field.getItem(path.peek().getVertical(), path.peek().
    ↪ getHorizontal()).getType() == ItemType.MARKER) {
91              path.pop();
92              try {
93                  makeMove(path.peek().getVertical(), path.peek().
    ↪ getHorizontal());
94              } catch (Exception x) {
95                  setBotBarrier();
96              }
97              return;
98          }
99          makeMove(path.peek().getVertical(), path.peek().
    ↪ getHorizontal());
100     }
101
102     private void setBotBarrier() throws FieldItemException {
103
104         if (!TOP.isActive()) {
105             setBotMarker();
106             return;
107         }
108
109         Stack<Coordinates> topPath = field.getPathToRow(TOP.marker.
```

```
    ↪ getCoordinates(), TOP.destinationRow);
110         Coordinates between = new Coordinates((topPath.peek().
    ↪ getVertical() + TOP.marker.getCoordinates().getVertical()) /
    ↪  2,
111                 (topPath.peek().getHorizontal() + TOP.marker.
    ↪ getCoordinates().getHorizontal()) / 2);
112         double rand = Math.random();
113         try {
114             if (rand < 0.5) {
115                 makeMove(between.getVertical() % 2 == 0 ? between.
    ↪ getVertical() - 1 : between.getVertical(),
116                         between.getHorizontal() % 2 == 0 ? between.
    ↪ getHorizontal() - 1 : between.getHorizontal(),
117                         rand > 0.15 ? BarrierPosition.HORIZONTAL :
    ↪ BarrierPosition.VERTICAL);
118             } else {
119                 makeMove(between.getVertical() % 2 == 0 ? between.
    ↪ getVertical() + 1 : between.getVertical(),
120                         between.getHorizontal() % 2 == 0 ? between.
    ↪ getHorizontal() + 1 : between.getHorizontal(),
121                         rand > 0.65 ? BarrierPosition.HORIZONTAL :
    ↪ BarrierPosition.VERTICAL);
122             }
123         } catch (Exception e) {
124             setBotMarker();
125         }
126
127     }
128
129     public void makeMove(int vertical, int horizontal) throws
    ↪ FieldItemException {
130
131         checkPlace(vertical, horizontal);
132         setItem(vertical, horizontal);
133     }
134
135     public void makeMove(int vertical, int horizontal,
    ↪ BarrierPosition position) throws FieldItemException,
    ↪ NoBarriersException {
136
137         //TODO зачем называть метод makeMove, когда совершается
    ↪ placeBarrier это( имя метода из Quoridor)
138
139         if (barriersNumber == 0) {
140             throw new NoBarriersException("you␣have␣no␣barriers");
    ↪          //TODO можно ещё передать у кого из игроков
141         }
142
143         checkPlace(vertical, horizontal, position);
144         setItem(vertical, horizontal, position);
145     }
146
147     private void checkPlace(int vertical, int horizontal) throws
    ↪ FieldItemException {
148
149         try {
150             field.getItem(vertical, horizontal);
151         } catch (ArrayIndexOutOfBoundsException e) {
152             throw new FieldBoundsException("impossible␣to␣place␣
    ↪ marker␣on␣" + vertical + "␣" + horizontal);
153         }
154
```

```
155         if (marker.getCoordinates().equals(new Coordinates(vertical
    ↪ , horizontal))) {
156             throw new ImpossibleToSetItemException("impossible to
    ↪ move to the same cell");
157         }
158
159         if (field.getColor(vertical, horizontal) == CellColor.WHITE
    ↪ ) {
160             throw new ImpossibleToSetItemException("impossible to
    ↪ set marker on white cell");
161         }
162
163         if (field.getItem(vertical, horizontal).getType() !=
    ↪ ItemType.EMPTY) {
164             throw new CellIsNotEmptyException("cell is not empty");
165         }
166
167         if (Coordinates.pathBetween(marker.getCoordinates(), new
    ↪ Coordinates(vertical, horizontal)) > 2.1) {
168
169             if (!jumpOverMarker(vertical, horizontal)) {
170                 throw new TooLongDistanceException("you can move
    ↪ just nearby cells");
171             }
172         }
173
174         if (field.getItem((marker.getCoordinates().getVertical() +
    ↪ vertical) / 2,
175                 (marker.getCoordinates().getHorizontal() +
    ↪ horizontal) / 2).getType() == ItemType.BARRIER) {
176             throw new ImpossibleToSetItemException("impossible to
    ↪ jump over the barrier");
177         }
178     }
179
180     private boolean jumpOverMarker(int vertical, int horizontal)
    ↪ throws FieldItemException {
181
182         // если прыгают "" прямо:
183         if (Coordinates.pathBetween(marker.getCoordinates(), new
    ↪ Coordinates(vertical, horizontal)) < 4.01 &&
184                 Coordinates.pathBetween(marker.getCoordinates(),
    ↪ new Coordinates(vertical,horizontal)) > 3.99) {
185
186             return jumpForward(vertical, horizontal);
187         }
188
189         // если прыгают "" по диагонали:
190         if (Coordinates.pathBetween(marker.getCoordinates(), new
    ↪ Coordinates(vertical, horizontal)) < 2.83 &&
191                 Coordinates.pathBetween(marker.getCoordinates(),
    ↪ new Coordinates(vertical,horizontal)) > 2.81) {
192
193             return jumpDiagonal(vertical, horizontal);
194         }
195
196         return false;
197     }
198
199     private boolean jumpForward(int vertical, int horizontal)
    ↪ throws FieldItemException {
200
```

```
201         Coordinates midCoordinates = new Coordinates( (marker.
    ↪ getCoordinates().getVertical() + vertical) / 2,
202                 (marker.getCoordinates().getHorizontal() +
    ↪ horizontal) / 2);
203
204         if (field.getItem(midCoordinates.getVertical(),
    ↪ midCoordinates.getHorizontal()).getType() == ItemType.MARKER
    ↪ ) {
205
206             if ( (field.getItem( (midCoordinates.getVertical() +
    ↪ marker.getCoordinates().getVertical()) / 2,
207                     (midCoordinates.getHorizontal() + marker.
    ↪ getCoordinates().getHorizontal()) / 2).getType() == ItemType
    ↪ .BARRIER) ||
208                     (field.getItem( (midCoordinates.getVertical() +
    ↪  vertical) / 2,
209                     (midCoordinates.getHorizontal() + horizontal) /
    ↪  2).getType() == ItemType.BARRIER) ) {
210
211                 throw new ImpossibleToSetItemException("impossible␣
    ↪ to␣set␣marker␣because␣of␣barrier");
212             }
213
214             return true;
215         }
216
217         return false;
218     }
219
220     private boolean jumpDiagonal(int vertical, int horizontal)
    ↪ throws FieldItemException {
221
222         Coordinates opponentsMarker;
223
224         if (field.getItem(marker.getCoordinates().getVertical(),
    ↪ horizontal).getType() == ItemType.MARKER) {
225             opponentsMarker = new Coordinates(marker.getCoordinates
    ↪ ().getVertical(), horizontal);
226         } else if (field.getItem(vertical, marker.getCoordinates().
    ↪ getHorizontal()).getType() == ItemType.MARKER) {
227             opponentsMarker = new Coordinates(vertical, marker.
    ↪ getCoordinates().getHorizontal());
228         } else { return false; }
229
230         if ( (field.getItem( (opponentsMarker.getVertical() +
    ↪ marker.getCoordinates().getVertical()) / 2,
231                 (opponentsMarker.getHorizontal() + marker.
    ↪ getCoordinates().getHorizontal()) / 2).getType() == ItemType
    ↪ .BARRIER)
232                 ||
233                 (field.getItem( (opponentsMarker.getVertical() +
    ↪ vertical) / 2,
234                         (opponentsMarker.getHorizontal() +
    ↪ horizontal) / 2).getType() == ItemType.BARRIER)) {
235
236             throw new ImpossibleToSetItemException("impossible␣to␣
    ↪ set␣marker␣because␣of␣barrier");
237         }
238
239         return true;
240     }
241
```

```
242     private void checkPlace(int vertical, int horizontal,
    ↪ BarrierPosition position) throws FieldItemException {
243
244         //TODO я бы переименовал в checkPlaceForBarrier
245
246         if (vertical % 2 == 0 || horizontal % 2 == 0) {
247             throw new ImpossibleToSetItemException("impossible to
    ↪ set barrier here");
248         }
249
250         if (position == BarrierPosition.VERTICAL) {
    ↪             //todo чтото- сделать
251             for (int i = vertical - Barrier.length + 1; i <=
    ↪ vertical + Barrier.length - 1; i++) {
252                 try {
253                     if (field.getItem(i, horizontal).getType() !=
    ↪ ItemType.EMPTY) {
254                         throw new CellIsNotEmptyException("
    ↪ impossible to place barrier here");
255                     }
256                     if (field.getColor(i, horizontal) == CellColor.
    ↪ BLACK) {
257                         throw new ImpossibleToSetItemException("
    ↪ impossible to set barrier on black cell");
258                     }
259                 } catch (ArrayIndexOutOfBoundsException e) {
260                     throw new FieldBoundsException("impossible to
    ↪ place barrier here");
261                 }
262             }
263
264         } else if (position == BarrierPosition.HORIZONTAL) {
265             for (int i = horizontal - Barrier.length + 1; i <=
    ↪ horizontal + Barrier.length - 1; i++) {
266                 try {
267                     if (field.getItem(vertical, i).getType() !=
    ↪ ItemType.EMPTY) {
268                         throw new CellIsNotEmptyException("
    ↪ impossible to place barrier here");
269                     }
270                     if (field.getColor(vertical, i) == CellColor.
    ↪ BLACK) {
271                         throw new ImpossibleToSetItemException("
    ↪ impossible to set barrier on black cell");
272                     }
273                 } catch (ArrayIndexOutOfBoundsException e) {
274                     throw new FieldBoundsException("impossible to
    ↪ place barrier here");
275                 }
276             }
277         }
278
279         Barrier probableBarrier = new Barrier(vertical, horizontal,
    ↪  position);
280         field.setItem(probableBarrier);
281
282         if (TOP.isActive && field.getPathToRow(TOP.getMarker().
    ↪ getCoordinates(), 16).empty()) {
283             field.clearCells(probableBarrier.getCoordinates());
284             throw new ImpossibleToSetItemException("you can't place
    ↪ barrier here. Player is locked");
285         }
```

```
286             if (BOTTOM.isActive && field.getPathToRow(BOTTOM.getMarker
    ↪ ().getCoordinates(), 0).empty()) {
287                 field.clearCells(probableBarrier.getCoordinates());
288                 throw new ImpossibleToSetItemException("you can't place
    ↪ barrier here. Player is locked");
289             }
290 //             if (RIGHT.isActive && !field.getPathToRow(RIGHT.getMarker
    ↪ ().getCoordinates(), ??????????)) //todo изменить логику
291
292             field.clearCells(probableBarrier.getCoordinates());
293         }
294
295         private void setItem(int vertical, int horizontal) {
296
297             //TODO это можно было бы назвать setMarker(~)
298
299             field.setItem(new Empty(marker.getCoordinates().getVertical
    ↪ (), marker.getCoordinates().getHorizontal()));
300             marker.moveTo(vertical, horizontal);
301             field.setItem(marker);
302         }
303
304         private void setItem(int vertical, int horizontal,
    ↪ BarrierPosition position) {
305
306             //TODO два setItem с разными параметрами, по моему должны
    ↪ делать примерно одно и тоже, а у тебя при добавление
    ↪ BarrierPosition изменяется тип добавляемого элемента, предлагаю
    ↪ переименовать метод
307             Barrier barrier = new Barrier(vertical, horizontal,
    ↪ position);
308             field.setItem(barrier);
309             barriers.add(barrier);
310             barriersNumber--;
311         }
312
313 }
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor;


import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Empty;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.ItemType;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Marker;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Owner;

import java.util.EmptyStackException;
import java.util.Stack;

public class Fox {

    private Marker marker;
    private QuoridorField field;
    private Marker target;

    public Fox(QuoridorField field) {

        this.field = field;
        int x, y;
        do {
            x = (int) (Math.random() * 10) % 9 * 2;
            y = (int) (Math.random() * 10) % 9 * 2;
        } while (field.getItem(x, y).getType() != ItemType.EMPTY);
        marker = new Marker(x, y, Owner.FOX);
        field.setItem(marker);

        double rand = Math.random();
        System.out.println("fox rand: " + rand);
        if (rand > 0.5) {
            target = QuoridorPlayer.TOP.getMarker();
        } else {
            target = QuoridorPlayer.BOTTOM.getMarker();
        }
    }

    public boolean makeMove() {

        Coordinates c = getNextCoordinates();
        // не проверяем потому, что проверяется в getNextCoordinates();
        field.setItem(new Empty(marker.getCoordinates().getVertical
    ↪ (), marker.getCoordinates().getHorizontal()));
        marker.moveTo(c.getVertical(), c.getHorizontal());
        field.setItem(marker);

        return c.equals(target.getCoordinates());
    }

    public Marker getMarker() {
        return marker;
    }

    public Coordinates getTarget() {
        return target.getCoordinates();
    }

    private Coordinates getNextCoordinates() {

        Stack<Coordinates> path = field.getPath(marker.
    ↪ getCoordinates(), target.getCoordinates());
        Coordinates c = marker.getCoordinates();
```

```
61
62          try {
63              if (field.getItem(path.peek().getVertical(), path.peek
    ↪ ().getHorizontal()).getType() != ItemType.EMPTY
64                      && !path.peek().equals(target.getCoordinates())
    ↪ ) {
65                  path.pop();
66                  path.peek();
67              }
68              c = path.peek();
69          } catch (EmptyStackException e) {}
70
71          return c;
72      }
73  }
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor;


public class Coordinates {

    private int vertical;
    private int horizontal;

    public static double pathBetween(Coordinates coordinates1,
    ↪ Coordinates coordinates2) {

        //TODO конечно бред, но можно выделить private метод,
    ↪ возведения в квадрат, по крайней мере, понятность кода увеличится
        return Math.sqrt((coordinates1.getVertical() - coordinates2
    ↪ .getVertical()) *
                         (coordinates1.getVertical() - coordinates2
    ↪ .getVertical()) +
                         (coordinates1.getHorizontal() -
    ↪ coordinates2.getHorizontal()) *
                         (coordinates1.getHorizontal() -
    ↪ coordinates2.getHorizontal()));
    }

    public int getVertical() {
        return vertical;
    }

    public int getHorizontal() {
        return horizontal;
    }

    public Coordinates(int vertical, int horizontal) {
        this.horizontal = horizontal;
        this.vertical = vertical;
    }

    @Override
    public boolean equals(Object o) {    //TODO однобуквенное имя
    ↪ выглядит не очень
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;

        Coordinates that = (Coordinates) o;

        if (vertical != that.vertical) return false;  //TODO както-
    ↪ криво, предлагаю объединить в одно return условие;
        return horizontal == that.horizontal;
    }

    @Override
    public int hashCode() {
        int result = vertical;
        result = 31 * result + horizontal;
        return result;
    }
}
```

```
1 package ru.spbstu.icc.kspt.zhuikov.quoridor;
2
3
4 public enum CellColor {
5     BLACK,
6     WHITE
7 }
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.items;


abstract public class Item {

    protected ItemType type = ItemType.EMPTY;

    protected Owner owner = Owner.NOBODY;

    public Item() {}

    public ItemType getType() { return type; }

    public Owner getOwner() { return  owner; }
}
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.items;


public enum ItemType {

    MARKER,
    BARRIER,
    EMPTY
}
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.items;


import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;

abstract public class OneCellItem extends Item {

    protected Coordinates coordinates = new Coordinates(0, 0);

    public OneCellItem(int vertical, int horizontal) {
        coordinates = new Coordinates(vertical, horizontal);
    }

    public Coordinates getCoordinates() {     //TODO возвращять
    ↪ копию, хотя если её изменить класс Coordinates, то и так нормально
        return coordinates;
    }
}
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.items;


import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;

import java.util.ArrayList;
import java.util.List;

abstract public class ManyCellsItem extends Item {

    protected List<Coordinates> coordinates = new ArrayList<
    ↪ Coordinates>();

    public ManyCellsItem() { }

```

```
15      public List<Coordinates> getCoordinates() {          //TODO по
        ↪ моему, не правильно отдавать поле возвращять( нужно скорее копию)
16          return coordinates;
17      } //todo может return new ArrayList<Coordinates>.addAll(
        ↪ coordinates);
18  }
```

```
1   package ru.spbstu.icc.kspt.zhuikov.quoridor.items;
2
3
4   public class Empty extends OneCellItem {
5
6       public Empty(int vertical, int horizontal) {
7           super(vertical, horizontal);
8           this.type = ItemType.EMPTY;
9       }
10
11  }
```

```
1   package ru.spbstu.icc.kspt.zhuikov.quoridor.items;
2
3
4   import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
5
6   public class Barrier extends ManyCellsItem {
7
8       public final static int length = 2;
9
10      public Barrier(int vertical, int horizontal, BarrierPosition
        ↪ position) {
11  //          super();                                  //TODO зачем это
        ↪ делать? без( этого работает)
12          this.type = ItemType.BARRIER;
13
14          if (position == BarrierPosition.VERTICAL) {
15              for (int i = vertical - length + 1; i <= vertical +
        ↪ length - 1; i++) {
16                  this.coordinates.add(new Coordinates(i, horizontal)
        ↪ );
17              }
18          } else if (position == BarrierPosition.HORIZONTAL) {
19              for (int i = horizontal - length + 1; i <= horizontal +
        ↪  length - 1; i++) {
20                  this.coordinates.add(new Coordinates(vertical, i));
21              }
22          }
23
24  //          if (coordinates.size() != realLength) {
25  //              throw new IllegalArgumentException("too long barrier.
        ↪  Its size must be equals to " + length);
26  //          }
27
28
29      }
30
31  }
```

```
1   package ru.spbstu.icc.kspt.zhuikov.quoridor.items;
2
3
4   import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
```

```
5
6  public class Marker extends OneCellItem {
7
8      public Marker(int vertical, int horizontal) {
9          super(vertical, horizontal);
10         this.type = ItemType.MARKER;
11     }
12
13     public Marker(int vertical, int horizontal, Owner owner) {
14         this(vertical, horizontal);
15         this.owner = owner;
16     }
17
18     public void moveTo(int vertical, int horizontal) {
19         coordinates = new Coordinates(vertical, horizontal);
20     }
21 }
```

```
1  package ru.spbstu.icc.kspt.zhuikov.quoridor.items;
2
3
4  public enum BarrierPosition {
5
6      VERTICAL,
7      HORIZONTAL;
8
9      @Override
10     public String toString() {
11         return super.toString().toLowerCase();
12     }
13 }
```

```
1  package ru.spbstu.icc.kspt.zhuikov.quoridor.items;
2
3
4  public enum Owner {
5
6      TOP,
7      BOTTOM,
8      FOX,
9      RIGHT,
10     LEFT,
11     NOBODY
12 }
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;


public class FieldBoundsException extends
    ↪ ArrayIndexOutOfBoundsException {
    public FieldBoundsException(String s) {
        super(s);
    }
}
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;


public class FieldItemException extends Exception {    //TODO
    ↪ Исключение Пункт Области, возможно здесь есть смысл, но я не могу
    ↪ сказать, когда конкретно кидать это исключение

    public FieldItemException(String s) {
        super(s);
    }

}
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;


public class CellIsNotEmptyException extends FieldItemException {

    public CellIsNotEmptyException(String s) {
        super(s);
    }

}
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;


public class ImpossibleToSetItemException extends
    ↪ FieldItemException {  //TODO пожалуйста сделай так, чтобы
    ↪ исключение содержало больше информации о себе, нежели одну строчку

    public ImpossibleToSetItemException(String s) {
        super(s);
    }

}
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;


public class NoBarriersException extends Exception {

    public NoBarriersException(String s) {
        super(s);
    }

}
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;


public class NoWinnerException extends Exception {              //
    ↪ TODO я бы не стал делать исключение для рядовой ситуации
    public NoWinnerException(String s) {
        super(s);
    }
}
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;


public class TooLongDistanceException extends FieldItemException {

    public TooLongDistanceException(String s) {
        super(s);
    }

}
```

6.2 Консольное приложение

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.console;


import ru.spbstu.icc.kspt.zhuikov.quoridor.Quoridor;
import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Player;

import java.util.Scanner;

public class ConsoleGame {

    private static Quoridor game = new Quoridor(2, false);
    private static ConsoleDrawer drawer = new ConsoleDrawer(game);

    public static void main(String[] args) {

        ConsoleGame consoleGame = new ConsoleGame();
        consoleGame.launch();

    }

    public void launch() {

        Scanner in = new Scanner(System.in);
        drawer.drawPlayerInformation(Player.TOP);
        drawer.drawField();
        drawer.drawPlayerInformation(Player.BOTTOM);
        drawer.drawTurn();
        while (!game.isEnd()) {

            try {
                Command command = CommandReader.read(in.nextLine())
    ↪ ;
                switch (command.getCommandType()) {
                    case BARRIER:
                        game.placeBarrier(command.getCoordinates().
    ↪ getVertical(),
```

```
35                                                  command.getCoordinates().
    ↪ getHorizontal(),
36                                                  command.
    ↪ getBarrierPosition());
37                          break;
38                      case MARKER:
39                          game.moveMarker(command.getCoordinates().
    ↪ getVertical(),
40                                                  command.getCoordinates().
    ↪ getHorizontal());
41                          break;
42                      case HELP:
43                          drawer.drawHelp();
44                          break;
45                  }
46              } catch (Exception e) {
47                  System.out.println(e.getMessage());
48                  System.out.println();
49                  drawer.drawTurn();
50                  continue;
51              }
52
53              System.out.println();
54              drawer.drawPlayerInformation(Player.TOP);
55              drawer.drawField();
56              drawer.drawPlayerInformation(Player.BOTTOM);
57              drawer.drawTurn();
58          }
59
60          drawer.drawWinner();
61      }
62
63  }
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.console;


import ru.spbstu.icc.kspt.zhuikov.quoridor.CellColor;
import ru.spbstu.icc.kspt.zhuikov.quoridor.Quoridor;
import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
    ↪ NoWinnerException;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Owner;
import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Field;
import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Player;

public class ConsoleDrawer {

    private Quoridor game = null;

    public ConsoleDrawer(Quoridor game) {
        this.game = game;
    }

    public void drawField() {

        Field field = game.getField();
        for (int i = 0; i < field.getRealSize(); i++) {
            for (int j = 0; j < field.getRealSize(); j++) {
                switch (field.getCell(i, j).getType()) {
                    case EMPTY:
                        if (field.getCell(i, j).getColor() ==
    ↪ CellColor.BLACK) {
                            System.out.print("O");
                        } else {
                            System.out.print("␣");
                        }
                        break;
                    case BARRIER:
                        System.out.print("#");
                        break;
                    case MARKER:
                        if (field.getCell(i, j).getOwner() == Owner
    ↪ .BOTTOM) {
                            System.out.print("B");
                        } else if (field.getCell(i, j).getOwner()
    ↪ == Owner.TOP) {
                            System.out.print("T");
                        }
                }
            }
            System.out.println();
        }

    }

    public void drawPlayerInformation(Player player) {

        System.out.println(player.name() + "␣" + game.
    ↪ getPlayerInformation(player).getBarrierNumber());
    }

    public void drawTurn() {

        System.out.print(game.getCurrentPlayer().name() + "␣player'
    ↪ s␣turn:␣");
    }
```

```
57
58      public void drawWinner() {
59
60          try {
61              Player winner = game.getWinner();
62              System.out.println(winner + "␣player␣won!");
63          } catch (NoWinnerException e ) {
64              System.out.println("why␣did␣you␣call␣me?");
65          }
66      }
67
68      public void drawHelp() {
69
70          String help = "Quoridor␣game.␣Commands␣help.\n" +
71                  "␣␣␣Command␣examples␣(without␣\"[␣]\":\n" +
72                  "␣␣␣␣␣[marker␣2␣2]␣-␣moves␣current␣player's␣marker␣
    ↪ to␣[2␣2]␣position␣if␣it␣possible\n" +
73                  "␣␣␣␣␣[barrier␣5␣5␣vertical]␣-␣places␣current␣
    ↪ player's␣barrier␣to␣[5␣5]␣position␣" +
74                  "in␣vertical␣if␣it␣possible\n" +
75                  "␣␣␣␣␣[barrier␣5␣5␣horizontal]␣-␣places␣current␣
    ↪ player's␣barrier␣to␣[5␣5]␣position␣" +
76                  "in␣horizontal␣if␣it␣possible\n";
77
78          System.out.println(help);
79      }
80
81  }
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.console;


import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;

public class Command {

    private CommandType commandType;
    private Coordinates coordinates;
    private BarrierPosition barrierPosition;

    public Command(CommandType commandType) {
        this.commandType = commandType;
    }

    public CommandType getCommandType() {
        return commandType;
    }

    public Coordinates getCoordinates() {
        return coordinates;
    }

    public BarrierPosition getBarrierPosition() {
        return barrierPosition;
    }

    public void setCoordinates(Coordinates coordinates) {
        this.coordinates = coordinates;
    }

    public void setBarrierPosition(BarrierPosition barrierPosition)
    ↪ {
        this.barrierPosition = barrierPosition;
    }
}
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.console;


public enum CommandType {

    BARRIER,
    MARKER,
    HELP,
  //  HIDE_HELP

}
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.console;


import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;

public class CommandReader {

    public static Command read(String string) throws
    ↪ IllegalArgumentException {

        Command command;

        if (string.toLowerCase().matches("\\s*marker\\s+-?\\d+\\s
    ↪ +-?\\d+\\s*")) {
            command = new Command(CommandType.MARKER);
            String[] stringArray = string.split("\\s+");
            command.setCoordinates(new Coordinates(Integer.parseInt
    ↪ (stringArray[1]),
                                                   Integer.parseInt
    ↪ (stringArray[2])));
            return command;
        }

        if (string.toLowerCase().matches("\\s*barrier\\s+-?\\d+\\s
    ↪ +-?\\d+\\s+(horizontal|vertical)\\s*")) {
            command = new Command(CommandType.BARRIER);
            String[] stringArray = string.split("\\s+");
            command.setCoordinates(new Coordinates(Integer.parseInt
    ↪ (stringArray[1]),
                                                   Integer.parseInt
    ↪ (stringArray[2])));
            if (stringArray[3].equals("vertical")) {
                command.setBarrierPosition(BarrierPosition.VERTICAL
    ↪ );
            } else {
                command.setBarrierPosition(BarrierPosition.
    ↪ HORIZONTAL);
            }

            return command;
        }

        if (string.toLowerCase().matches("\\s*help\\s*")) {
            return new Command(CommandType.HELP);
        }

//          if (string.toLowerCase().matches("\\s*hide\\s+help\\s*"))
    ↪ {
//              return new Command(CommandType.HIDE_HELP);
//          }

        throw new IllegalArgumentException("unknown command");
    }

}
```

6.3 Графическое приложение

```
1  package ru.spbstu.icc.kspt.zhuikov.quoridor.gui;
2
3  import javax.swing.*;
4
5  class MainFrame extends JFrame {
6
7      MainFrame(String s) {
8          super(s);
9          setSize(640, 480);
10         add(new MenuPanel(this));
11         setResizable(false);
12         setVisible(true);
13         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14
15     }
16
17     public static void main(String[] args) {
18         SwingUtilities.invokeLater(new Runnable() {
19             public void run() {
20                 new MainFrame("Quoridor");
21             }
22         });
23     }
24 }
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.gui;


import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class MenuPanel extends JPanel {

    private JButton startOnePlayer = new JButton("1 Player");
    private JButton startTwoPlayer = new JButton("2 Player");
    private JButton settingsButton = new JButton("Settings");
    private JButton exitButton = new JButton("Exit");
    private MainFrame frame;
    private final Image bg = new ImageIcon("pictures/gamePics/
    ↪ background_menu1.jpg").getImage();

    MenuPanel(MainFrame frame) {

        this.frame = frame;
        setLayout(null);

        startOnePlayer.setLocation(225, 220);
        startOnePlayer.setSize(200, 30);
        startOnePlayer.addMouseListener(new StartListener(true));

        startTwoPlayer.setLocation(225, 270);
        startTwoPlayer.setSize(200, 30);
        startTwoPlayer.addMouseListener(new StartListener(false));

        settingsButton.setLocation(225, 320);
        settingsButton.setSize(200, 30);
        settingsButton.addMouseListener(new SettingsListener());


        exitButton.setLocation(225, 370);
        exitButton.setSize(200, 30);
        exitButton.addActionListener(new ExitListener());

        add(startOnePlayer);
        add(startTwoPlayer);
        add(settingsButton);
        add(exitButton);
    }

    @Override
    protected void paintComponent(Graphics g) {

        super.paintComponent(g);
        g.drawImage(bg, 0, 0, null);
    }

    private class StartListener implements MouseListener {

        private boolean bots = false;
        StartListener(boolean bots) {
            this.bots = bots;
        }

        @Override
        public void mouseClicked(MouseEvent e) {
            frame.setContentPane(new GamePanel(frame, bots));
```

```
62          }
63
64          @Override
65          public void mousePressed(MouseEvent e) {
66          }
67
68          @Override
69          public void mouseReleased(MouseEvent e) {
70              startOnePlayer.setText("d"); // ??????????????????????
71          }
72
73          @Override
74          public void mouseEntered(MouseEvent e) {
75          }
76
77          @Override
78          public void mouseExited(MouseEvent e) {
79
80          }
81      }
82
83      private class SettingsListener implements MouseListener {
84
85          @Override
86          public void mouseClicked(MouseEvent e) {
87              frame.setContentPane(new SettingsPanel(frame));
88          }
89
90          @Override
91          public void mousePressed(MouseEvent e) {
92
93          }
94
95          @Override
96          public void mouseReleased(MouseEvent e) {
97              settingsButton.setText("d");
98          }
99
100         @Override
101         public void mouseEntered(MouseEvent e) {
102
103         }
104
105         @Override
106         public void mouseExited(MouseEvent e) {
107
108         }
109     }
110
111     private class ExitListener implements ActionListener {
112
113         @Override
114         public void actionPerformed(ActionEvent e) {
115             System.exit(0);
116         }
117     }
118 }
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.gui;


import ru.spbstu.icc.kspt.zhuikov.quoridor.Quoridor;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class SettingsPanel extends JPanel {

    private final Image bg = new ImageIcon("pictures/gamePics/background_menu1.jpg").getImage();
    private MainFrame frame;
    private JButton backButton = new JButton("Back");
    private JSlider stepSlider = new JSlider(0, 40, Quoridor.getFoxTime());
    private JSlider frequencySlider = new JSlider(1, 20, Quoridor.getFoxFrequency());
    private JLabel stepLabel = new JLabel("");
    private JLabel frequencyLabel = new JLabel("");

    SettingsPanel(MainFrame frame) {

        this.frame = frame;
        setLayout(null);

        JLabel changeFoxTime = new JLabel("Fox spawn time (in steps from beginning): ");
        changeFoxTime.setLocation(130, 210);
        changeFoxTime.setSize(400, 20);
        changeFoxTime.setFont(new Font("Arial", Font.BOLD + Font.ITALIC, 15));

        stepSlider.setSize(300, 45);
        stepSlider.setFont(new Font("Arial",Font.BOLD + Font.ITALIC, 14));
        stepSlider.setBackground(new Color(210, 230, 250));
        stepSlider.setLocation(110, 240);
        stepSlider.setMajorTickSpacing(5);
        stepSlider.setPaintLabels(true);
        stepSlider.setPaintTicks(true);
        stepSlider.addChangeListener(e -> {
            stepLabel.setText("" + stepSlider.getValue());
            Quoridor.setFoxTime(stepSlider.getValue());
        });

        stepLabel = new JLabel("" + stepSlider.getValue());
        stepLabel.setSize(140, 45);
        stepLabel.setLocation(480, 242);
        stepLabel.setFont(new Font("Arial", Font.BOLD + Font.ITALIC, 16));

        JLabel changeFoxTurn = new JLabel("Fox move frequency (once in a set steps): ");
        changeFoxTurn.setLocation(130, 300);
        changeFoxTurn.setSize(400, 20);
        changeFoxTurn.setFont(new Font("Arial", Font.BOLD + Font.ITALIC, 15));

        frequencySlider.setSize(300, 45);
```

```
54          frequencySlider.setFont(new Font("Arial",Font.BOLD + Font.
        ↪ ITALIC, 14));
55          frequencySlider.setBackground(new Color(210, 230, 250));
56          frequencySlider.setLocation(110, 330);
57          frequencySlider.setMajorTickSpacing(2);
58          frequencySlider.setPaintLabels(true);
59          frequencySlider.setPaintTicks(true);
60          frequencySlider.addChangeListener(e -> {
61              frequencyLabel.setText("" + frequencySlider.getValue())
        ↪ ;
62              Quoridor.setFoxFrequency(frequencySlider.getValue());
63          });
64
65          frequencyLabel = new JLabel("" + frequencySlider.getValue()
        ↪ );
66          frequencyLabel.setSize(140, 45);
67          frequencyLabel.setLocation(480, 332);
68          frequencyLabel.setFont(new Font("Arial", Font.BOLD + Font.
        ↪ ITALIC, 16));
69
70          backButton.setLocation(200, 400);
71          backButton.setSize(200, 30);
72          backButton.addMouseListener(new BackListener());
73
74          add(changeFoxTime);
75          add(stepLabel);
76          add(stepSlider);
77          add(changeFoxTurn);
78          add(frequencyLabel);
79          add(frequencySlider);
80          add(backButton);
81      }
82
83      @Override
84      protected void paintComponent(Graphics g) {
85
86          super.paintComponent(g);
87          g.drawImage(bg, 0, 0, null);
88      }
89
90      private class BackListener implements MouseListener {
91
92          @Override
93          public void mouseClicked(MouseEvent e) {
94              frame.setContentPane(new MenuPanel(frame));
95          }
96
97          @Override
98          public void mousePressed(MouseEvent e) {
99
100         }
101
102         @Override
103         public void mouseReleased(MouseEvent e) {
104             backButton.setText("d");
105         }
106
107         @Override
108         public void mouseEntered(MouseEvent e) {
109
110         }
111
```

```
112             @Override
113             public void mouseExited(MouseEvent e) {
114
115             }
116         }
117
118 }
```

```java
package ru.spbstu.icc.kspt.zhuikov.quoridor.gui;

import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
import ru.spbstu.icc.kspt.zhuikov.quoridor.Quoridor;
import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
    ↪ FieldItemException;
import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
    ↪ NoBarriersException;
import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
    ↪ NoWinnerException;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.ItemType;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Owner;
import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Cell;
import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Field;
import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Player;

import javax.swing.*;
import javax.swing.border.LineBorder;
import java.awt.*;
import java.awt.event.*;

class GamePanel extends JPanel {

    private final Color bottomColor = Color.red;
    private final Color topColor = Color.blue;

    private MainFrame frame;
    private Quoridor game;
    private FieldPanel fieldPanel;
    private BarrierPanel barrierPanel;
    private JLabel statusLabel;
    private FoxPanel foxPanel;
    private Image bg = new ImageIcon("pictures/gamePics/
    ↪ background_game.jpg").getImage();

    GamePanel(MainFrame frame, boolean bots) {

        this.frame = frame;
        game = new Quoridor(2, bots);

        statusLabel = new JLabel("fff");
        statusLabel.setSize(470, 25);
        statusLabel.setLocation(45, 395);
        statusLabel.setBorder(new LineBorder(Color.BLACK, 2));
        statusLabel.setFont(new Font("Arial", Font.ITALIC + Font.
    ↪ BOLD, 17));

        updateStatusLabel();
        add(statusLabel);

        JButton menuButton = new JButton("Menu");
        menuButton.setLocation(435, 20);
        menuButton.setSize(140, 20);
        menuButton.addActionListener(new MenuListener());
        add(menuButton);

        fieldPanel = new FieldPanel();
        barrierPanel = new BarrierPanel();
        foxPanel = new FoxPanel();

        add(foxPanel);
```

```
58          add(fieldPanel);
59          add(barrierPanel);
60
61          setBackground(new Color(200, 200, 200));
62          setLayout(null);
63          setVisible(true);
64
65      }
66
67      private void updateStatusLabel() {
68          if (game.getCurrentPlayer().name().equals("TOP")) {
69              statusLabel.setText("Blue player's turn");
70          } else if (game.getCurrentPlayer().name().equals("BOTTOM"))
    ↪     {
71              statusLabel.setText("Red player's turn");
72          }
73      }
74
75      @Override
76      protected void paintComponent(Graphics g) {
77
78          super.paintComponent(g);
79          g.drawImage(bg, 0, 0, null);
80      }
81
82      private class MenuListener implements ActionListener {
83
84          @Override
85          public void actionPerformed(ActionEvent e) {
86              fieldPanel.setVisible(false);
87              barrierPanel.setVisible(false);
88              frame.setContentPane(new MenuPanel(frame));
89          }
90      }
91
92      private class FieldPanel extends JPanel {
93
94          private Field field;
95          private final int cellSize = 30;
96          private final int spaceSize = 10;
97          private boolean pickedMarker = false;
98          private boolean pickedBarrier = false;
99          private BarrierPosition barrierPosition;
100         private Coordinates pickedMarkerCoordinates;
101         private FieldMouseListener fieldMouseListener;
102
103         FieldPanel() {
104
105             this.field = game.getField();
106
107             setLayout(null);
108             setBackground(Color.white);
109             setLocation(25, 30);
110             setSize(350, 350);
111             setVisible(true);
112
113             fieldMouseListener = new FieldMouseListener();
114             addMouseListener(fieldMouseListener);
115         }
116
117         void pickBarrier(BarrierPosition position) {
118             pickedMarker = false;
```

```
119             this.repaint();
120             pickedBarrier = true;
121             barrierPosition = position;
122             statusLabel.setText("Place " + position + " barrier...");
123         }
124
125         private void unpickBarrier(int x, int y) {
126
127             Coordinates coordinates = roundToOdd(x, y);
128             try {
129                 game.placeBarrier(coordinates.getVertical(), coordinates.getHorizontal(), barrierPosition);
130                 pickedBarrier = false;
131                 updateStatusLabel();
132                 barrierPanel.updateText();
133             } catch (FieldItemException | NoBarriersException e)  {
134                 statusLabel.setText(e.getMessage());
135                 pickedBarrier = false;
136                 Timer timer = new Timer(1500, new ActionListener() {
137                     @Override
138                     public void actionPerformed(ActionEvent e) {
139                         updateStatusLabel();
140                     }
141                 });
142                 timer.setRepeats(false);
143                 timer.start();
144               //  barrierPanel.updateText();
145             }
146         }
147
148         private Coordinates roundToOdd(int x, int y) {
149
150             Coordinates initCoordinates = convertCoordinates(x, y);
151             int newX = x;
152             int newY = y;
153             if (initCoordinates.getVertical() % 2 == 0) {
154                 int leftBound  = (initCoordinates.getVertical() / 2) * (cellSize + spaceSize) - spaceSize / 2; // в пикселах
155                 int rightBound = (initCoordinates.getVertical() / 2 + 1) * (cellSize + spaceSize) - spaceSize / 2;
156                 if (Math.abs(y - leftBound) < Math.abs(y - rightBound)) {
157                     newY = leftBound;
158                 } else {
159                     newY = rightBound;
160                 }
161             }
162
163             if (initCoordinates.getHorizontal() % 2 == 0) {
164                 int leftBound  = (initCoordinates.getHorizontal() / 2) * (cellSize + spaceSize) - spaceSize / 2; // в пикселах
165                 int rightBound = (initCoordinates.getHorizontal() / 2 + 1) * (cellSize + spaceSize) - spaceSize / 2;
166                 if (Math.abs(x - leftBound) < Math.abs(x - rightBound)) {
167                     newX = leftBound;
168                 } else {
169                     newX = rightBound;
170                 }
171             }
```

```
172
173             if (newX > this.getSize().width) newX = this.getSize().
    ↪ width - cellSize - spaceSize / 2;
174             if (newX < 0) newX = cellSize + spaceSize / 2;
175             if (newY > this.getSize().height) newY = this.getSize()
    ↪ .height - cellSize - spaceSize / 2;
176             if (newY < 0) newY = cellSize + spaceSize / 2;
177
178             return convertCoordinates(newX, newY);
179         }
180
181         @Override
182         public void paint(Graphics g) {
183             super.paint(g);
184
185             g.setColor(new Color(170, 170, 170));
186
187             for (int i = 0; i <= field.getSize(); i++ ) {
188 //              g.drawLine(i * (cellSize + spaceSize), 0, i * (
    ↪ cellSize + spaceSize), 350);
189 //              g.drawLine(i * (cellSize + spaceSize) - spaceSize, 0,
    ↪  i * (cellSize + spaceSize) - spaceSize, 350);
190 //
191 //              g.drawLine(0, i * (cellSize + spaceSize), 350, i * (
    ↪ cellSize + spaceSize));
192 //              g.drawLine(0, i * (cellSize + spaceSize) - spaceSize,
    ↪  350, i * (cellSize + spaceSize) - spaceSize);
193
194                 for (int j = 0; j <= field.getSize(); j++) {
195                     g.fillRect(i * (cellSize + spaceSize), j * (
    ↪ cellSize + spaceSize), cellSize, cellSize);
196                 }
197             }
198
199             this.field = game.getField();
200
201             for (int i = 0; i < field.getRealSize(); i++) {
202                 for (int j = 0; j < field.getRealSize(); j++) {
203                     Cell cell = field.getCell(i, j);
204                     switch (cell.getType()) {
205                         case MARKER: {
206                             if (cell.getOwner() == Owner.TOP) {
207                                 g.setColor(topColor);
208                                 g.fillOval(j * (cellSize +
    ↪ spaceSize) / 2 + cellSize / 8,
209                                         i * (cellSize + spaceSize)
    ↪ / 2 + cellSize / 8,
210                                         24, 24);
211                             } else if (cell.getOwner() == Owner.
    ↪ BOTTOM) {
212                                 g.setColor(bottomColor);
213                                 g.fillOval(j * (cellSize +
    ↪ spaceSize) / 2 + cellSize / 8,
214                                         i * (cellSize + spaceSize)
    ↪ / 2 + cellSize / 8,
215                                         24, 24);
216                             } else if (cell.getOwner() == Owner.FOX
    ↪ ) {
217                                 g.setColor(new Color(255, 77, 0));
218                                 g.fillPolygon(new int [] {j * (
    ↪ cellSize + spaceSize) / 2 + 3, j * (cellSize + spaceSize) /
    ↪ 2 + cellSize / 2, j * (cellSize + spaceSize) / 2 + cellSize
```

```
    ↪ - 3 },
219                                         new int [] {i * (cellSize +
    ↪  spaceSize) / 2 + 3, i * (cellSize + spaceSize) / 2 +
    ↪ cellSize - 3, i * (cellSize + spaceSize) / 2 + 3}, 3);
220                         }
221                         break;
222                     }
223                     case BARRIER: {
224                         g.setColor(new Color(140, 90, 90));
225                         if (i % 2 == 1 && j % 2 == 1)
226                             g.fillRect((j + 1) * (cellSize +
    ↪ spaceSize) / 2 - spaceSize,
227                                     (i + 1) * (cellSize +
    ↪ spaceSize) / 2 - spaceSize, spaceSize, spaceSize);
228                         if (i % 2 == 1 && j % 2 == 0)
229                             g.fillRect(j * (cellSize +
    ↪ spaceSize) / 2,
230                                     (i + 1) * (cellSize +
    ↪ spaceSize) / 2 - spaceSize, cellSize, spaceSize);
231                         if (i % 2 == 0 && j % 2 == 1)
232                             g.fillRect((j + 1) * (cellSize +
    ↪ spaceSize) / 2 - spaceSize,
233                                     i * (cellSize + spaceSize)
    ↪ / 2, spaceSize, cellSize);
234                         break;
235                     }
236                 }
237             }
238         }
239
240         foxPanel.updateLabel();
241
242         if (pickedMarker) {
243             g.setColor(Color.BLACK);
244             Graphics2D g2 = (Graphics2D) g;
245             g2.setStroke(new BasicStroke(3));
246             g2.drawRect(pickedMarkerCoordinates.getHorizontal()
    ↪  * (cellSize + spaceSize) / 2,
247                     pickedMarkerCoordinates.getVertical() * (
    ↪ cellSize + spaceSize) / 2, cellSize, cellSize);
248             for (Coordinates c : game.getPossibleMoves()) {
249                 g.setColor(Color.green);
250                 g.fillRect(c.getHorizontal() * (cellSize +
    ↪ spaceSize) / 2,
251                         c.getVertical() * (cellSize + spaceSize
    ↪ ) / 2, cellSize, cellSize);
252             }
253         }
254
255         try {
256             if (game.isEnd()) {
257                 this.removeMouseListener(fieldMouseListener);
258                 barrierPanel.removeListeners();
259                 if (game.getWinner().name().equals("TOP")) {
260                     statusLabel.setText("Blue player won!");
261                 } else if (game.getWinner().name().equals("
    ↪ BOTTOM")) {
262                     statusLabel.setText("Red player won!");
263                 } else if (game.getWinner().name().equals("FOX"
    ↪ )) {
264                     statusLabel.setText("Fox won!");
265                 }
```

```
266                 }
267             } catch (NoWinnerException e) {}
268
269         }
270
271         /**
272          * Переводит "" из координат курсора в координаты поля игры
273          * @return координаты клетки поля, на которую кликнули
274          */
275         private Coordinates convertCoordinates(int x, int y) {
276
277             int fieldVertical;       // координаты на игровом поле (0 -
    ↪  realSize)
278             int fieldHorizontal;
279
280             if (y % (cellSize + spaceSize) > cellSize) {
281                 fieldVertical = y / (cellSize + spaceSize) * 2 + 1;
282             } else {
283                 fieldVertical = y / (cellSize + spaceSize) * 2;
284             }
285
286             if (x % (cellSize + spaceSize) > cellSize) {
287                 fieldHorizontal = x / (cellSize + spaceSize) * 2 +
    ↪ 1;
288             } else {
289                 fieldHorizontal = x / (cellSize + spaceSize) * 2;
290             }
291
292             return new Coordinates(fieldVertical, fieldHorizontal);
293         }
294
295         private boolean onCell(int x, int y) {
296
297             Coordinates fieldCoordinates = convertCoordinates(x, y)
    ↪ ;
298
299             return fieldCoordinates.getVertical() % 2 == 0 &&
    ↪ fieldCoordinates.getHorizontal() % 2 == 0;
300         }
301
302         private void pickMarker(Cell marker) {
303
304             pickedMarker = marker.getOwner().name().equals(game.
    ↪ getCurrentPlayer().name());
305             pickedMarkerCoordinates = new Coordinates(marker.
    ↪ getVertical(), marker.getHorizontal());
306         }
307
308         private void unpickMarker(Coordinates newCoordinates) {
309
310             pickedMarker = false;
311             try {
312                 game.moveMarker(newCoordinates.getVertical(),
    ↪ newCoordinates.getHorizontal());
313                 updateStatusLabel();
314                 barrierPanel.updateText();
315             } catch (FieldItemException | NoBarriersException e ) {
316                 statusLabel.setText(e.getMessage());
317                 Timer timer = new Timer(1500, e1 ->
    ↪ updateStatusLabel());
318                 timer.setRepeats(false);
319                 timer.start();
```

```
320             }
321         }
322 
323         private class FieldMouseListener implements MouseListener {
324 
325             public void mouseClicked(MouseEvent e) {
326                 if (onCell(e.getX(), e.getY())) {
327                     Coordinates fieldCoordinates =
    ↪ convertCoordinates(e.getX(), e.getY());
328                     Cell cell = field.getCell(fieldCoordinates.
    ↪ getVertical(), fieldCoordinates.getHorizontal());
329                     if (cell.getType() == ItemType.MARKER) {
330                         pickMarker(cell);
331                     } else if (pickedMarker) {
332                         unpickMarker(fieldCoordinates);
333                     }
334                 } else if (pickedBarrier) {
335                     unpickBarrier(e.getX(), e.getY());
336                 }
337                 repaint();
338             }
339 
340             public void mousePressed(MouseEvent e) {
341 
342             }
343 
344             public void mouseReleased(MouseEvent e) {
345             }
346 
347             public void mouseEntered(MouseEvent e) {
348             }
349 
350             public void mouseExited(MouseEvent e) {
351 
352             }
353         }
354     }
355 
356     private class BarrierPanel extends JPanel {
357 
358         private JLabel bottomBarriersNumber = new JLabel();
359         private JLabel topBarriersNumber = new JLabel();
360         private JButton barrierButton = new JButton("Place␣barrier"
    ↪ );
361         private JButton cancelButton = new JButton("Cancel");
362         private BarrierListener barrierListener;
363         private CancelBarrierListener cancelBarrierListener;
364         private JRadioButton horizontal = new JRadioButton("
    ↪ Horizontal", true);
365         private JRadioButton vertical = new JRadioButton("Vertical"
    ↪ , false);
366 
367         BarrierPanel() {
368 
369             Color color = new Color(190, 140, 140);
370             setLayout(null);
371 
372             setBorder(new LineBorder(Color.BLACK, 2));
373             setLocation(410, 180);
374             setSize(190, 200);
375             setBackground(color);
376             setVisible(true);
```

```
377
378             bottomBarriersNumber.setLocation(15, 10);
379             bottomBarriersNumber.setSize(185, 20);
380             bottomBarriersNumber.setFont(new Font("Arial", Font.
    ↪ ITALIC + Font.BOLD, 14));
381             topBarriersNumber.setLocation(10, 30);
382             topBarriersNumber.setSize(185, 20);
383             topBarriersNumber.setFont(new Font("Arial", Font.BOLD +
    ↪  Font.ITALIC, 14));
384
385             updateText();
386
387             barrierButton.setLocation(25, 70);
388             barrierButton.setSize(140, 20);
389             barrierListener = new BarrierListener();
390             barrierButton.addMouseListener(barrierListener);
391
392             cancelButton.setLocation(25, 160);
393             cancelButton.setSize(140, 20);
394             cancelBarrierListener = new CancelBarrierListener();
395             cancelButton.addMouseListener(cancelBarrierListener);
396
397             ButtonGroup group = new ButtonGroup();
398             horizontal.setSize(100, 20);
399             horizontal.setLocation(45, 100);
400             horizontal.setBackground(color);
401             group.add(horizontal);
402
403             vertical.setSize(100, 20);
404             vertical.setLocation(45, 130);
405             vertical.setBackground(color);
406             group.add(vertical);
407
408             add(bottomBarriersNumber);
409             add(topBarriersNumber);
410             add(barrierButton);
411             add(cancelButton);
412             add(vertical);
413             add(horizontal);
414
415         }
416
417         void removeListeners() {
418             barrierButton.removeMouseListener(barrierListener);
419             cancelButton.removeMouseListener(cancelBarrierListener)
    ↪ ;
420         }
421
422         void updateText() {
423             bottomBarriersNumber.setText("Red␣has␣" + game.
    ↪ getPlayerInformation(Player.BOTTOM).getBarrierNumber() + "␣
    ↪ barriers");
424             topBarriersNumber.setText("Blue␣has␣" + game.
    ↪ getPlayerInformation(Player.TOP).getBarrierNumber() + "␣
    ↪ barriers");
425         }
426
427         private class BarrierListener implements MouseListener {
428
429             @Override
430             public void mouseClicked(MouseEvent e) {
431                 fieldPanel.pickBarrier(horizontal.isSelected() ?
```

```
    ↪ BarrierPosition.HORIZONTAL : BarrierPosition.VERTICAL);
432             }
433
434             @Override
435             public void mousePressed(MouseEvent e) {
436
437             }
438
439             @Override
440             public void mouseReleased(MouseEvent e) {
441
442             }
443
444             @Override
445             public void mouseEntered(MouseEvent e) {
446
447             }
448
449             @Override
450             public void mouseExited(MouseEvent e) {
451
452             }
453         }
454
455         private class CancelBarrierListener implements
    ↪ MouseListener {
456             @Override
457             public void mouseClicked(MouseEvent e) {
458                 fieldPanel.pickedBarrier = false;
459                 updateStatusLabel();
460             }
461
462             @Override
463             public void mousePressed(MouseEvent e) {
464
465             }
466
467             @Override
468             public void mouseReleased(MouseEvent e) {
469
470             }
471
472             @Override
473             public void mouseEntered(MouseEvent e) {
474
475             }
476
477             @Override
478             public void mouseExited(MouseEvent e) {
479
480             }
481         }
482     }
483
484
485     class FoxPanel extends JPanel {
486
487         private JLabel foxAppearingLabel;
488         private JLabel foxFrequencyLabel;
489
490         FoxPanel() {
491
```

```
492             setLayout(null);
493             setLocation(410, 60);
494             setSize(190, 100);
495             setBorder(new LineBorder(Color.BLACK, 2));
496             setBackground(new Color(190, 140, 140));
497 
498             foxAppearingLabel = new JLabel();
499             foxAppearingLabel.setSize(180, 20);
500             foxAppearingLabel.setLocation(7, 5);
501             foxAppearingLabel.setFont(new Font("Arial", Font.ITALIC
    ↪  + Font.BOLD, 14));
502 
503             JLabel foxFrequencyTextLabel = new JLabel("The␣Fox␣
    ↪ moves␣once␣");
504             foxFrequencyTextLabel.setSize(180, 20);
505             foxFrequencyTextLabel.setLocation(7, 40);
506             foxFrequencyTextLabel.setFont(new Font("Arial", Font.
    ↪ ITALIC + Font.BOLD, 14));
507 
508             foxFrequencyLabel = new JLabel("in␣" + Quoridor.
    ↪ getFoxFrequency() + "␣steps");
509             foxFrequencyLabel.setSize(100, 20);
510             foxFrequencyLabel.setLocation(100, 70);
511             foxFrequencyLabel.setFont(new Font("Arial", Font.ITALIC
    ↪  + Font.BOLD, 14));
512 
513             add(foxAppearingLabel);
514             add(foxFrequencyTextLabel);
515             add(foxFrequencyLabel);
516 
517         }
518 
519         void updateLabel() {
520             if (Quoridor.getFoxTime() - game.getStep() > 0) {
521                 foxAppearingLabel.setText("The␣Fox␣appears␣" + (
    ↪ Quoridor.getFoxTime() - game.getStep()));
522             } else {
523                 foxAppearingLabel.setLocation(26, 5);
524                 foxAppearingLabel.setText("The␣Fox␣is␣here!");
525             }
526         }
527     }
528 }
```

6.4 Модульные тесты

```
1  package ru.sbpstu.icc.kspt.Zhuikov.courseWork;
2  // todo пусть тесты покрывают всю логику
3
4  import org.junit.Test;
5  import ru.spbstu.icc.kspt.zhuikov.quoridor.QuoridorField;
6  import ru.spbstu.icc.kspt.zhuikov.quoridor.QuoridorPlayer;
7  import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.*;
8  import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Barrier;
9  import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;
10 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.ItemType;
11
12 import static org.junit.Assert.assertEquals;
13
14 public class BarrierTest {
15
16     @Test
17     public void testBarrierSet() throws FieldItemException,
   ↪ NoBarriersException {
18
19         QuoridorField field = new QuoridorField(9);
20         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
21         player.createPlayer(field, false);
22
23         player.makeMove(5, 3, BarrierPosition.VERTICAL);
24
25         assertEquals(ItemType.BARRIER, field.getItem(5, 3).getType
   ↪ ());
26         assertEquals(ItemType.BARRIER, field.getItem(4, 3).getType
   ↪ ());
27         assertEquals(ItemType.BARRIER, field.getItem(6, 3).getType
   ↪ ());
28
29         player.makeMove(13, 11, BarrierPosition.HORIZONTAL);
30
31         assertEquals(ItemType.BARRIER, field.getItem(13, 10).
   ↪ getType());
32         assertEquals(ItemType.BARRIER, field.getItem(13, 11).
   ↪ getType());
33         assertEquals(ItemType.BARRIER, field.getItem(13, 12).
   ↪ getType());
34     }
35
36
37     @Test(expected = ImpossibleToSetItemException.class)
38     public void testBlackCellSet() throws FieldItemException,
   ↪ NoBarriersException {
39
40         QuoridorField field = new QuoridorField(9);
41         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
42         player.createPlayer(field, false);
43         player.makeMove(2, 8, BarrierPosition.HORIZONTAL);
44     }
45
46     @Test(expected = ImpossibleToSetItemException.class)
47     public void testSetBetweenBlackCells() throws
   ↪ FieldItemException, NoBarriersException {
48
49         QuoridorField field = new QuoridorField(9);
50         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
51         player.createPlayer(field, false);
```

```
52          player.makeMove(7, 12, BarrierPosition.VERTICAL);
53      }
54
55  //      @Test(expected = FieldBoundsException.class)
56  //      public void testWrongCoordinates() throws FieldItemException,
        ↪  NoBarriersException {
57  //
58  //          QuoridorField field = new QuoridorField(9);
59  //          QuoridorPlayer player = QuoridorPlayer.BOTTOM;
60  //          player.createPlayer(field, false);
61  //          player.makeMove(0, 1, BarrierPosition.VERTICAL);
62  //      }
63
64      @Test(expected = CellIsNotEmptyException.class)
65      public void testImpossibleSet() throws FieldItemException,
        ↪ NoBarriersException {
66
67          QuoridorField field = new QuoridorField(9);
68          QuoridorPlayer player = QuoridorPlayer.BOTTOM;
69          player.createPlayer(field, false);
70
71          player.makeMove(5, 3, BarrierPosition.VERTICAL);
72          player.makeMove(3, 3, BarrierPosition.VERTICAL);
73      }
74
75      @Test (expected = NoBarriersException.class)
76      public void testNoBarriers() throws FieldItemException,
        ↪ NoBarriersException {
77
78          QuoridorField field = new QuoridorField(9);
79          QuoridorPlayer player = QuoridorPlayer.TOP;
80          player.createPlayer(field, false);
81
82          for (int i = 1; i < 16; i+=2) {
83              player.makeMove(1, i, BarrierPosition.VERTICAL); // 8
        ↪ barriers
84          }
85          player.makeMove(5, 1, BarrierPosition.VERTICAL);
86          player.makeMove(5, 3, BarrierPosition.VERTICAL);
87
88          player.makeMove(5, 5, BarrierPosition.VERTICAL);
89      }
90
91      @Test (expected = ImpossibleToSetItemException.class)
92      public void testPlayerBlock() throws FieldItemException,
        ↪ NoBarriersException {
93
94          QuoridorField field = new QuoridorField(9);
95          QuoridorPlayer bottom = QuoridorPlayer.BOTTOM;
96          QuoridorPlayer top = QuoridorPlayer.TOP;
97          bottom.createPlayer(field, false);
98          top.createPlayer(field, false);
99
100         field.setItem(new Barrier(1, 7, BarrierPosition.VERTICAL));
101         field.setItem(new Barrier(1, 9, BarrierPosition.VERTICAL));
102
103         bottom.makeMove(3, 8, BarrierPosition.HORIZONTAL);
104     }
105 }
```

```java
package ru.sbpstu.icc.kspt.Zhuikov.courseWork;


import org.junit.Test;
import ru.spbstu.icc.kspt.zhuikov.quoridor.CellColor;
import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
import ru.spbstu.icc.kspt.zhuikov.quoridor.QuoridorField;
import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
    ↪ FieldItemException;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Barrier;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;

import static org.junit.Assert.assertEquals;

public class FieldTest {

    @Test
    public void testInitialization() {

        QuoridorField field = new QuoridorField(9);

        assertEquals(CellColor.BLACK, field.getColor(0, 0));
        assertEquals(CellColor.WHITE, field.getColor(0, 1));
        assertEquals(CellColor.WHITE, field.getColor(1, 1));
        assertEquals(CellColor.BLACK, field.getColor(2, 2));
    }

    @Test
    public void testPathBetweenCells1() {  // пустое поле

        QuoridorField field = new QuoridorField(9);
        assertEquals(false, field.getPathToRow(new Coordinates(16,
    ↪ 8), 0).empty());
        assertEquals(false, field.getPathToRow(new Coordinates(0,
    ↪ 8), 10).empty());
    }

    @Test
    public void testPathBetweenCells2() throws FieldItemException {
    ↪  // перегородка"" по вертикали

        QuoridorField field = new QuoridorField(9);

        for (int i = 1; i <= 13; i+=4) {
            field.setItem(new Barrier(i, 7, BarrierPosition.
    ↪ VERTICAL));
        }
        field.setItem(new Barrier(15, 7, BarrierPosition.HORIZONTAL
    ↪ ));
        field.setItem(new Barrier(15, 9, BarrierPosition.VERTICAL))
    ↪ ;

        assertEquals(false, field.getPathToRow(new Coordinates(16,
    ↪ 8), 0).empty());
        assertEquals(false, field.getPathToRow(new Coordinates(0,
    ↪ 16), 0).empty());
        assertEquals(false, field.getPathToRow(new Coordinates(0,
    ↪ 16), 16).empty());
    }

    @Test
    public void testPathBetweenCells3() throws FieldItemException {
```

```
    ↪  // перегородка "" по горизонтали
53
54          QuoridorField field = new QuoridorField(9);
55          for (int i = 1; i <= 13; i+=4) {
56              field.setItem(new Barrier(7, i, BarrierPosition.
    ↪ HORIZONTAL));
57          }
58          field.setItem(new Barrier(7, 15, BarrierPosition.VERTICAL))
    ↪ ;
59          field.setItem(new Barrier(5, 15, BarrierPosition.HORIZONTAL
    ↪ ));
60
61          assertEquals(false, field.getPathToRow(new Coordinates(0,
    ↪ 4), 2).empty());
62          assertEquals(true, field.getPathToRow(new Coordinates(0, 4)
    ↪ , 10).empty());
63          assertEquals(true, field.getPathToRow(new Coordinates(16,
    ↪ 6), 0).empty());
64      }
65
66      @Test
67      public void testPathBetweenCells4() throws FieldItemException {
    ↪  // закрытая фишка
68
69          QuoridorField field = new QuoridorField(9);
70          field.setItem(new Barrier(15, 7, BarrierPosition.VERTICAL))
    ↪ ;
71          field.setItem(new Barrier(15, 9, BarrierPosition.VERTICAL))
    ↪ ;
72          field.setItem(new Barrier(13, 8, BarrierPosition.HORIZONTAL
    ↪ ));
73
74          assertEquals(true, field.getPathToRow(new Coordinates(16,
    ↪ 8), 6).empty());
75          assertEquals(false, field.getPathToRow(new Coordinates(16,
    ↪ 8), 16).empty());
76          assertEquals(false, field.getPathToRow(new Coordinates(0,
    ↪ 8), 16).empty());
77      }
78
79  }
```

```java
package ru.sbpstu.icc.kspt.Zhuikov.courseWork;


import org.junit.Test;

import ru.spbstu.icc.kspt.zhuikov.quoridor.QuoridorField;
import ru.spbstu.icc.kspt.zhuikov.quoridor.QuoridorPlayer;
import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.*;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Barrier;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.ItemType;
import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Marker;

import static org.junit.Assert.assertEquals;

public class MarkerTest {

    @Test
    public void testInitialCoordinates() {

        QuoridorField field = new QuoridorField(9);
        QuoridorPlayer player1 = QuoridorPlayer.TOP;
        QuoridorPlayer player2 = QuoridorPlayer.BOTTOM;
        player1.createPlayer(field, false);
        player2.createPlayer(field, false);

        assertEquals(ItemType.MARKER, field.getItem(0, 8).getType()
    ↪ );
        assertEquals(ItemType.MARKER, field.getItem(16, 8).getType
    ↪ ());
    }

    @Test
    public void testMoving() throws FieldItemException {

        QuoridorField field = new QuoridorField(9);
        QuoridorPlayer player = QuoridorPlayer.BOTTOM;
        player.createPlayer(field, false);

        player.makeMove(14, 8);

        assertEquals(ItemType.MARKER, field.getItem(14, 8).getType
    ↪ ());

    }

    @Test (expected = FieldBoundsException.class)
    public void testMovingOutOfBounds() throws FieldItemException {

        QuoridorField field = new QuoridorField(9);
        QuoridorPlayer player = QuoridorPlayer.BOTTOM;
        player.createPlayer(field, false);

        player.makeMove(18, 8);

    }

    @Test (expected = TooLongDistanceException.class)
    public void testFarMoving() throws FieldItemException {

        QuoridorField field = new QuoridorField(9);
        QuoridorPlayer player = QuoridorPlayer.BOTTOM;
```

```
60          player.createPlayer(field, false);
61
62          player.makeMove(14, 6);
63      }
64
65      @Test (expected = ImpossibleToSetItemException.class)
66      public void testImpossibleMoving() throws FieldItemException {
67
68          QuoridorField field = new QuoridorField(9);
69          QuoridorPlayer player = QuoridorPlayer.BOTTOM;
70          player.createPlayer(field, false);
71
72          player.makeMove(15, 8);
73      }
74
75      @Test (expected = ImpossibleToSetItemException.class)
76      public void testMovingToMarkerCell() throws FieldItemException
        ↪ {
77
78          QuoridorField field = new QuoridorField(9);
79          QuoridorPlayer player = QuoridorPlayer.BOTTOM;
80          player.createPlayer(field, false);
81
82          player.makeMove(16, 8);
83      }
84
85      @Test (expected = ImpossibleToSetItemException.class)
86      public void testJumpOverBarrier() throws FieldItemException {
87
88          QuoridorField field = new QuoridorField(9);
89          QuoridorPlayer player = QuoridorPlayer.BOTTOM;
90          player.createPlayer(field, false);
91          Barrier barrier = new Barrier(15, 8, BarrierPosition.
        ↪ HORIZONTAL);
92          field.setItem(barrier);
93
94          player.makeMove(14, 8);
95      }
96
97      @Test
98      public void testJumpOverMarker_Forward() throws
        ↪ FieldItemException {
99
100         QuoridorField field = new QuoridorField(9);
101         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
102         player.createPlayer(field, false);
103         field.setItem(new Marker(14, 8));
104
105         player.makeMove(12, 8);
106         assertEquals(ItemType.MARKER, field.getItem(12, 8).getType
        ↪ ());
107     }
108
109     @Test
110     public void testJumpOverMarker_Diagonal() throws
        ↪ FieldItemException {
111
112         QuoridorField field = new QuoridorField(9);
113         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
114         player.createPlayer(field, false);
115         field.setItem(new Marker(14, 8));
116
```

```
117             player.makeMove(14, 10);
118             assertEquals(ItemType.MARKER, field.getItem(14, 10).getType
        ↪ ());
119         }
120
121         @Test (expected = ImpossibleToSetItemException.class)
122         public void testImpossibleJumpOverMarker_Forward() throws
        ↪ FieldItemException {
123
124             QuoridorField field = new QuoridorField(9);
125             QuoridorPlayer player = QuoridorPlayer.BOTTOM;
126             player.createPlayer(field, false);
127             field.setItem(new Marker(14, 8));
128             field.setItem(new Barrier(13, 8, BarrierPosition.HORIZONTAL
        ↪ ));
129
130             player.makeMove(12, 8);
131         }
132
133         @Test (expected = ImpossibleToSetItemException.class)
134         public void testImpossibleJumpOverMarker_Diagonal() throws
        ↪ FieldItemException {
135
136             QuoridorField field = new QuoridorField(9);
137             QuoridorPlayer player = QuoridorPlayer.BOTTOM;
138             player.createPlayer(field, false);
139             field.setItem(new Marker(14, 8));
140             field.setItem(new Barrier(15, 9, BarrierPosition.VERTICAL))
        ↪ ;
141
142             player.makeMove(14, 10);
143         }
144 }
```