

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"	2
1.1	Краткое описание игры	2
1.2	Правила игры	2
1.3	Задание	3
1.4	Дополнение игры.	3
1.5	Диаграмма прецедентов использования	3
1.6	Вывод	4
2	Проектирование приложения, реализующего игру "Коридор"	5
2.1	Проектирование библиотеки	5
2.2	Вывод	6
3	Реализация игры "Коридор"	6
3.1	Среда разработки	6
3.2	Реализация консольного приложения	6
3.3	Реализация библиотеки	8
3.4	Реализация графического приложения	10
3.5	Вывод	14
4	Процесс обеспечения качества и тестирование приложения "Коридор"	14
4.1	Просмотр кода	14
4.2	Проведенные демонстрации	14
4.3	Тестирование кода приложения	15
4.4	Вывод	15
5	Вывод	15
6	Приложение 1. Листинги кода	16
6.1	Библиотека	16
6.2	Консольное приложение	41
6.3	Графическое приложение	47
6.4	Модульные тесты	63

1 Настольная игра "Коридор"

С древнейших времен люди играли в игры. История настольных игр насчитывает 5500 лет. В современных исследованиях настольных игр высказывается мнение, что их история была общей. Так, Дэвид Парлетт, автор "Оксфордской истории настольных игр" считает, что все, даже самые современные игры имеют древнейшие прототипы и, следовательно, общие корни.¹

В современном мире игры также популярны. В число настольных игр входят игры со специальным полем, карточные игры и другие. Отдельный вид игр - компьютерные игры. Они подразделяются на несколько жанров: стратегии, логические, аркады и другие. Целью курсовой работы является разработка приложения, позволяющего играть в логическую настольную игру "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 можно либо передвинуть фишку с целью быстрого достижения финиша, либо установить перегородку, которые могут помешать достичь финиша противнику. В игре "Коридор" поле общее, поэтому любая поставленная перегородка может помешать обоим игрокам.

Ниже приведены выбранные для реализации правила игры ².

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.

¹David Parlett The Oxford History of Board Games. Oxford Press, 1999.
ISBN 0-19-212998-8

²www.gaga.ru/gaga/files/pdf/rules/230.pdf

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 программно реализованного игрока. 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

Были составлены диаграммы прецедентов использования.

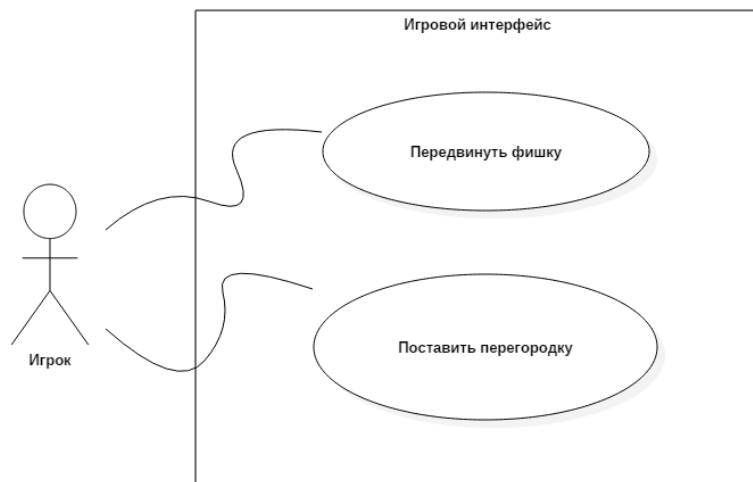


Рис. 1: Диаграмма прецедентов использования.

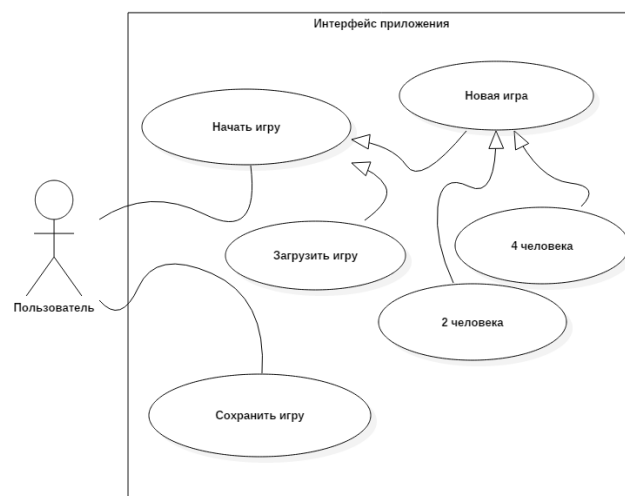


Рис. 2: Диаграмма прецедентов использования.

На рисунках 1 и 2 представлены диаграммы прецедентов использования.

1.6 Вывод

Определены правила игры "Коридор", которую планируется реализовать. Составлены диаграммы прецедентов использования приложения.

2 Проектирование приложения, реализующего игру "Коридор"

Приложение должно позволять играть в игру двум игрокам, а также против программно реализованного игрока.

2.1 Проектирование библиотеки

Библиотека - ядро приложения. Здесь содержатся основные классы, необходимые для представления игры. Для создания графического приложения была выбрана библиотека Swing.

В API выделены следующие сигнатуры:

- *public void moveMarker(int vertical, int horizontal) throws FieldItemException, NoBarriersException* - метод, передвигающий фишку текущего игрока в заданные координаты поля. (Текущий игрок - игрок, чья очередь делать ход.)
- *public void placeBarrier(int vertical, int horizontal, BarrierPosition position) throws FieldItemException, NoBarriersException* - метод, размещающий перегородку текущего игрока на клетку с заданными координатами и в заданном направлении (горизонтально или вертикально). При этом перегородка занимает три клетки.
- *public List<Coordinates> getPossibleMoves()* - метод, возвращающий список доступных ходов фишкой текущего игрока.
- *public Player getPlayerInformation(Player player)* - метод, позволяющий получить информацию о заданном игроке.
- *public Player getCurrentPlayer()* - метод, возвращающий текущего игрока.
- *public boolean isEnd()* - метод, возвращающий *true*, если какой-либо игрок достиг своей финишной линии и *false* в обратном случае. Также возвращается *true*, если the Fox "ловит" фишку какого-либо игрока.
- *public static int getFoxTime()* и *public static int getFoxFrequency()* - методы, возвращающие информацию о the Fox: ход, на котором появляется the Fox, и частота, с которой the Fox делает ход соответственно.

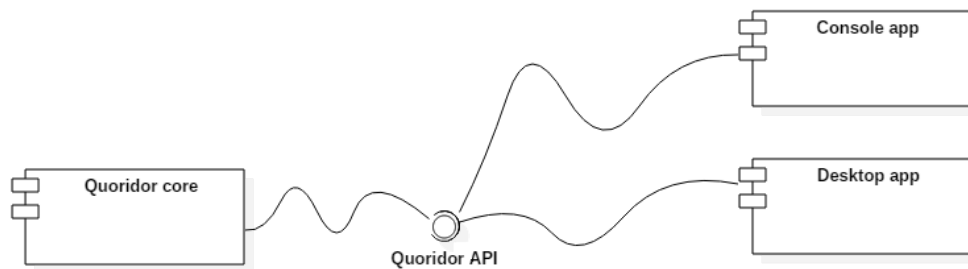


Рис. 3: Диаграмма компонентов.

На рисунке 3 приведена диаграмма компонентов разрабатываемого приложения.

2.2 Вывод

Были определены основные методы API приложения, выбрана библиотека для реализации графического приложения. Также составлена диаграмма компонентов.

3 Реализация игры "Коридор"

3.1 Среда разработки

Интегрированная среда разработки IntelliJ IDEA 2016.2.5.

Язык: Java 1.8.

Система автоматической сборки: Gradle 2.14.

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 соответствующее сообщение об ошибке.

На рисунках 4 и 5 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 0 B 0 0 0 0
BOTTOM 10
TOP player's turn: barrier 7 7 Horizontal

```

Рис. 4: 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 0 0 0 0 0

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

```

Рис. 5: Состояние игры после выполнения команды *barrier 7 7 Horizontal*.

На рисунке 5 представлено изменение поля игры - появление барьера на указанной в команде позиции.

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 (ТОР, БОТТОМ...). 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.

В классе имеется флаг *isBot*. По умолчанию он содержит значение *false*. Для создания игрока, управляемого "компьютером", необходимо присвоить флагу значение *true*. После этого, игрок будет совершать ходы без участия пользователя: с вероятностью 50% будет передвинута фишка или поставлена перегородка. Перед передвижением фишки вычисляется кратчайший путь до линии финиша и затем делается необходимый ход. При установке барьера вычисляется кратчайший для противника путь к финишной линии и ставится перегородка, преграждающая следующий наилучший ход. Вычисление пути происходит по алгоритму поиска в ширину на графе, представляющем собой клетки поля, соединенные ребрами при отсутствии перегородки между ними.

- 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

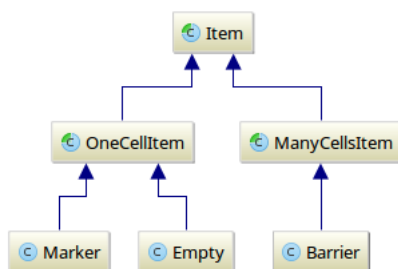


Рис. 6: Диаграмма классов, реализующих объекты на поле.

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, используемые в библиотеке приложения, содержатся в подпакете *.exceptions*. Иерархия исключений показана на рисунке 7.

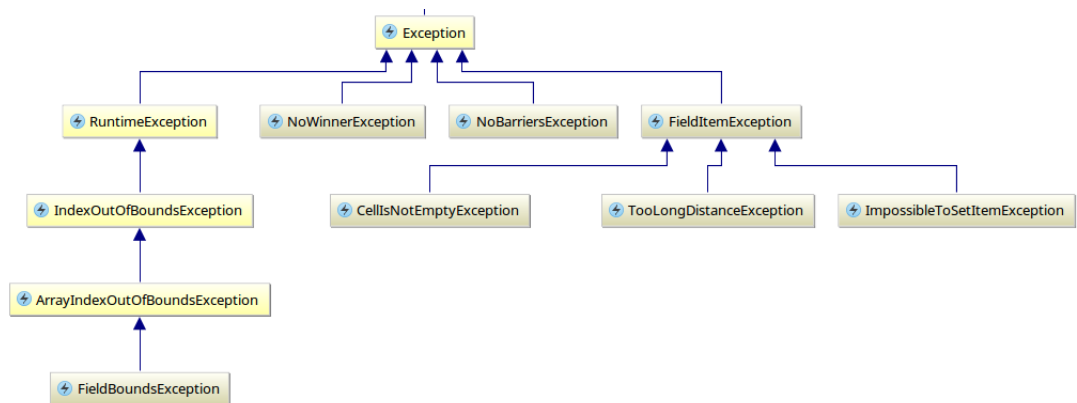


Рис. 7: Диаграмма классов, реализующих исключения.

Диаграммы показывают, что при реализации библиотеки использовалось наследование. Наследование сделало код более расширяемым.

3.4 Реализация графического приложения

Для создания графического приложения была выбрана библиотека Swing.

На рисунке 8 представлено главное окно приложения. Пользователю, предоставляется возможность начать одиночную игру против программно реализованного игрока, начать игру для двух человек, изменить настройки игры или выйти.



Рис. 8: Главное меню графического приложения.

На рисунке 9 - 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. Присутствует 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.

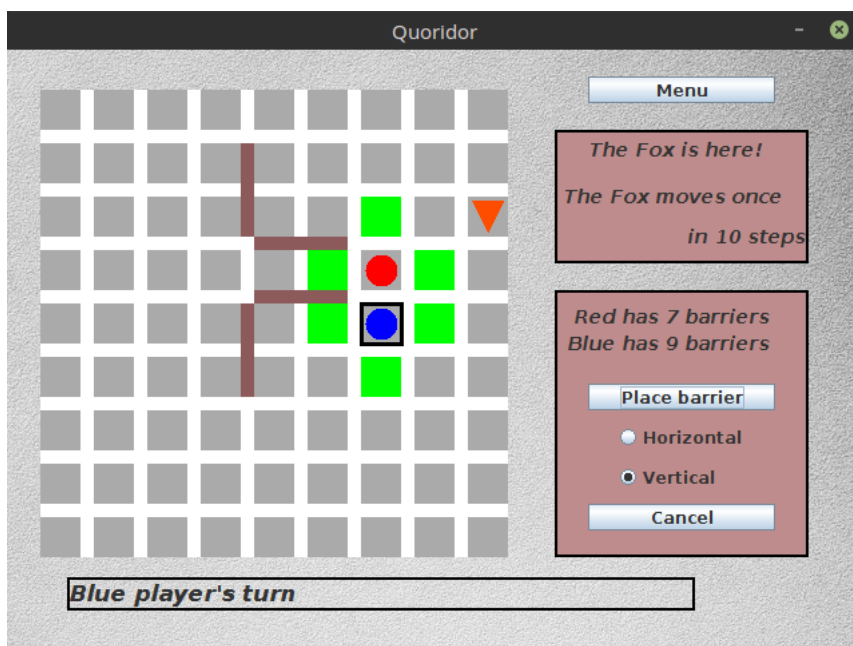


Рис. 9: Скриншот окна игры. Зеленым цветом подсвечиваются ходы, на которые можно сходить фишкой текущему игроку.

На рисунке 10 представлена победа игрока the Fox.

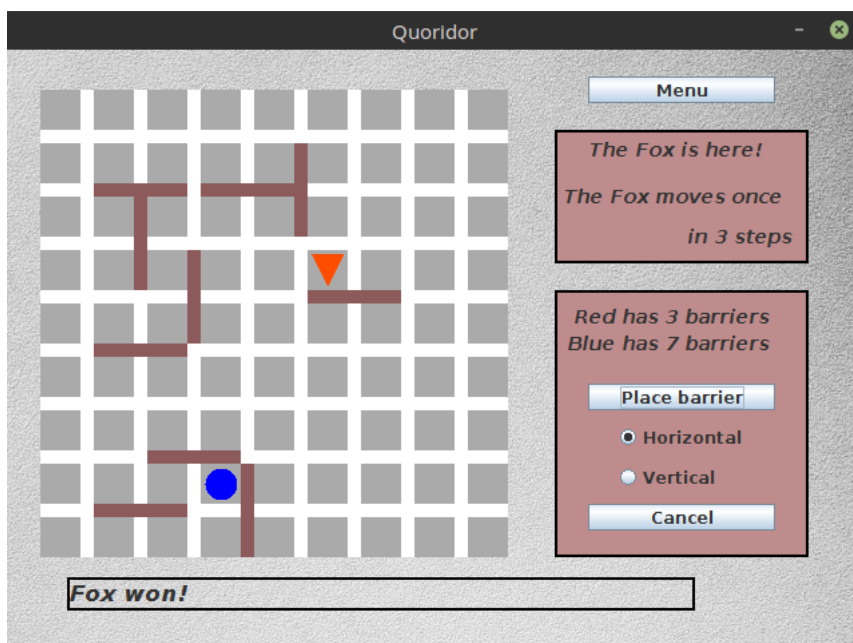


Рис. 10: Победа игрока the Fox: удалось догнать красного игрока.

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

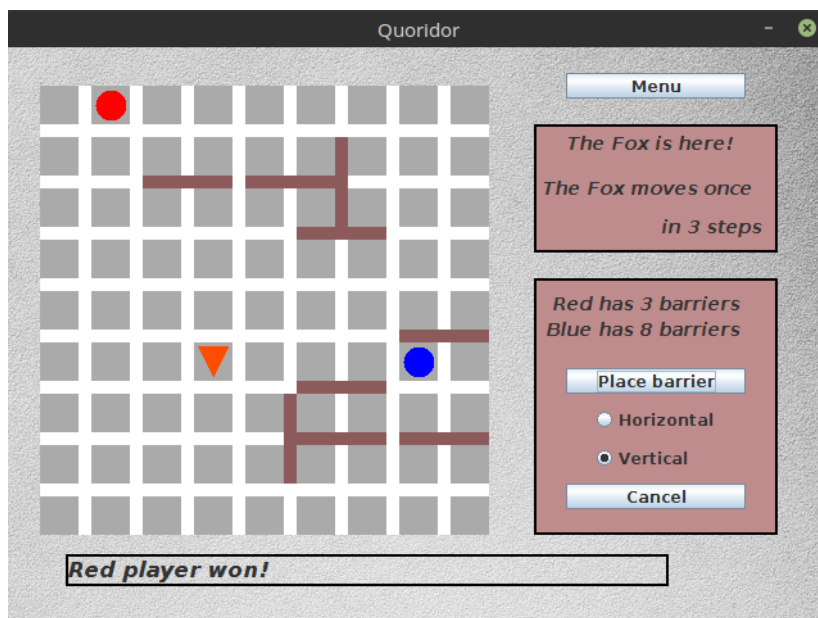


Рис. 11: Красный игрок одержал победу, достигнув финишной линии.

На рисунке 12 представлен скриншот настроек приложения.

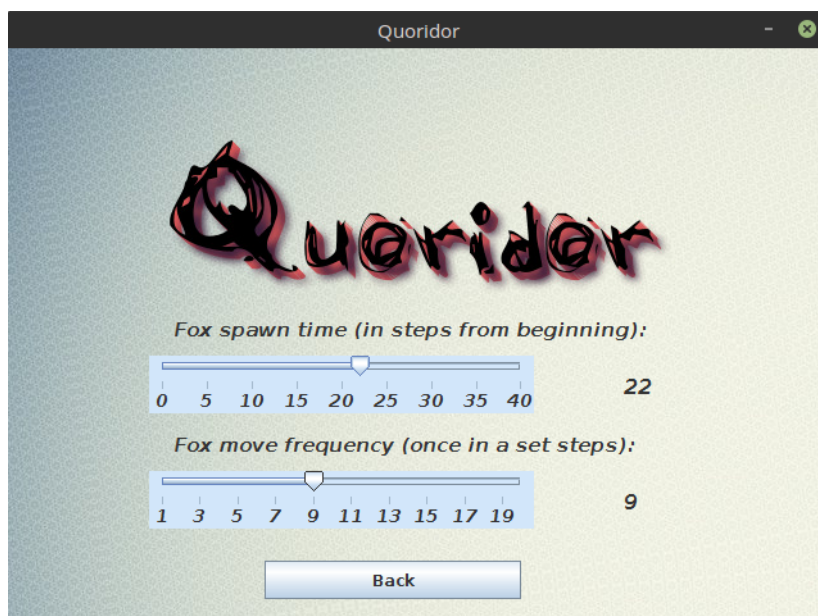


Рис. 12: Настройки игры.

В настройках можно указать некоторые параметры игры: время появления the Fox (в шагах от начала игры) и частоту хода the Fox (один раз за указанное количество шагов). Шаг игры - ход одного игрока.

Таким образом, разработано графическое приложение, позволяющее играть в игру "Коридор" как одному игроку, так и двум.

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 приложения "Коридор"

4.1 Просмотр кода

В ходе проектирования была проведена проверка исходного кода программы с целью обнаружения и исправления ошибок (code review). Проверка была проведена 10 декабря 2016 года Мальцевым Михаилом. В результате было получено около 30-ти замечаний.³

Большая часть замечаний исправлена.

4.2 Проведенные демонстрации

Была проведена одна презентация приложения (20 декабря 2016 года). Получены следующие 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 переноса перегородки курсором мыши, учтены и исправлены.

³<https://github.com/Zhuikov/Notwithstanding/commit/caa42f5cfb1b4a905356f7ba53201a2d904d3cb9>

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 `QuoridorPlayer`: проверяется установка перегородки на поле, возможность перемещения фишки на позиции. С помощью возможностей среды разработки было подсчитано покрытие кода тестами. Результаты представлены на рисунке 13.

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. 13: Процент покрытия кода тестами.

4.4 Вывод

Разработанное приложение тестировалось модульными тестами. Проводилась проверка кода программы, в результате которой были найдены некоторые недочеты проектирования приложения и другие ошибки. Демонстрация приложения позволила расширить уже существующую функциональность игры.

5 Вывод

В результате работы над проектом была разработана библиотека для игры "Коридор" по выбранным правилам с некоторым дополнением. Код приложения протестирован модульными тестами, а также просмотрен коллегой на наличие каких-либо недочетов. Посредством библиотеки Swing было создано приложение, предоставляющее пользователю функциональность ядра и позволяющее играть в игру "Коридор" одному игроку против программно реализованного игрока или двум игрокам друг против друга.

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 пакет( 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(
58 ↪ currentPlayer));
59 }
60 public static void setFoxTime(int foxTime) {
61     if (foxTime < 0) {
62         throw new IllegalArgumentException("Fox_time_must_be_>=
63 ↪ 0");
64     }
65     Quoridor.foxTime = foxTime;
66 }
67 public static int getFoxTime() {
68     return foxTime;
69 }
70 }
71 public static int getFoxFrequency() {
72     return foxFrequency;
73 }
74 }
75 public static void setFoxFrequency(int foxTurn) {
76     if (foxTurn < 1 ) {
77         throw new IllegalArgumentException("Fox_Frequency_must_
78 ↪ be_>0");
79     }
80     Quoridor.foxFrequency = foxTurn;
81 }
82 }
83 public int getStep() {
84     return step;
85 }
86 }
87 public Field getField() {
88     return new Field(field);
89 }
90 }
91 //todo по рукам бы надавать наверно надо за такое...
92 public Player getPlayerInformation(Player player) {
93     switch (player) {
94         case TOP:
95             player.createPlayer(QuoridorPlayer.TOP.
96 ↪ getBarriersNumber());
97             return player;
98         case BOTTOM:
99             player.createPlayer(QuoridorPlayer.BOTTOM.
100 ↪ getBarriersNumber());
101             return player;
102     }
103     //         case RIGHT:
104     //             player.createPlayer(QuoridorPlayer.RIGHT.
105 ↪ getBarriersNumber());
106     //             return player;

```

```

106 //          default:
107 //          player.createPlayer(QuoridorPlayer.LEFT.
    ↪ getBarriersNumber());
108 //          return player;
109     }
110     throw new AssertionError("unknown_player" + player);
111 }
112
113 public boolean isEnd() { //TODO возможно следует
    ↪ подумать об использовании шаблона Наблюдатель
114     if (QuoridorPlayer.TOP.getMarker().getCoordinates().
    ↪ getVertical() == QuoridorPlayer.TOP.getDestinationRow()) {
115         return true;
116     }
117
118     if (QuoridorPlayer.BOTTOM.getMarker().getCoordinates().
    ↪ getVertical() == QuoridorPlayer.BOTTOM.getDestinationRow())
119     ↪ {
120         return true;
121     }
122
123     if (fox != null && fox.getMarker().getCoordinates().equals(
    ↪ fox.getTarget())) {
124         return true;
125     }
126
127     // todo тут еще потом других сделать
128     return false;
129 }
130
131 public Player getWinner() throws NoWinnerException { //TODO
    ↪ по моему-, при использовании Наблюдателя метод атрофируется
132
133     if (isEnd()) {
134         if (QuoridorPlayer.TOP.getMarker().getCoordinates().
    ↪ getVertical() == field.getRealSize() - 1) {
135             return Player.TOP;
136         }
137         if (QuoridorPlayer.BOTTOM.getMarker().getCoordinates().
    ↪ getVertical() == 0) {
138             return Player.BOTTOM;
139         }
140         if (fox != null && fox.getMarker().getCoordinates().
    ↪ equals(fox.getTarget())) {
141             return Player.FOX;
142         }
143     }
144
145     // todo вообще ужас! за это точно надо надавать...
146     throw new NoWinnerException("There_is_no_winner");
147 }
148
149 public void moveMarker(int vertical, int horizontal)
150     throws FieldItemException, NoBarriersException { //
    ↪ TODO странное название у исключения, возможно есть смысл
    ↪ переименовать в Выход за границу поля, но это не точно
151
152     players.get(currentPlayer).makeMove(vertical, horizontal);
153     changePlayerTurn();
154 }
155

```

```

156 public void placeBarrier(int vertical, int horizontal,
157 ↪ BarrierPosition position)
158     throws FieldItemException, NoBarriersException {
159     players.get(currentPlayer).makeMove(vertical, horizontal,
160 ↪ position);
161     changePlayerTurn();
162 }
163 public List<Coordinates> getPossibleMoves() {
164     return players.get(currentPlayer).getPossibleMoves();
165 }
166 private void changePlayerTurn() throws FieldItemException,
167 ↪ NoBarriersException {
168     if (++currentPlayer == players.size()) {
169         currentPlayer = 0;
170     }
171     if (isEnd()) return;
172     if (fox != null && step % foxFrequency == 0) {
173         fox.makeMove();
174         if (isEnd()) return;
175     }
176     if (step == foxTime) {
177         fox = new Fox(field);
178     }
179     step++;
180     if (players.get(currentPlayer).isBot()) {
181         players.get(currentPlayer).makeBotMove();
182         if (isEnd()) return;
183         changePlayerTurn();
184     }
185 }
186 }
187 }
188 }
189 }
190 }
191 }
192 }
193 }
194 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor;
2
3
4 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.*;
5
6 import java.util.*;
7
8 public class QuoridorField {
9     //TODO я бы засунул этот класс в пакет связанный с полем да( и вообще
10    ↪ всё, что связано с полем засунул в один пакет)
11
12    private final int realSize;
13    private final int size;
14    private Cell[][] field; //TODO почему бы не использовать,
15    ↪ какуюнибудь коллекцию? конечно( банально, но например, List<List
16    ↪ <~>>)
17
18    public QuoridorField(int size) {
19
20        this.size = size;
21        realSize = size * 2 - 1;
22        field = new Cell[realSize][realSize];
23        for (int i = 0; i <= realSize - 1; i++) {
24            for (int j = 0; j <= realSize - 1; j++) {
25                if ((i % 2 == 0) && (j % 2 == 0)) {
26                    field[i][j] = new Cell(CellColor.BLACK, new
27    ↪ Empty(i, j));
28                } else {
29                    field[i][j] = new Cell(CellColor.WHITE, new
30    ↪ Empty(i, j));
31                }
32            }
33        }
34    }
35
36    public QuoridorField() {
37        this(9);
38    }
39
40    private class Cell {
41        private CellColor color;
42        private Item item;
43
44        Cell(CellColor color, Item item) {
45            this.color = color;
46            this.item = item;
47        }
48    }
49
50    public void setItem(OneCellItem item) {
51        field[item.getCoordinates().getVertical()][item.
52    ↪ getCoordinates().getHorizontal()].item = item;
53    }
54
55    public void setItem(ManyCellsItem item) {
56        for (Coordinates coordinates : item.getCoordinates()) {
57            field[coordinates.getVertical()][coordinates.
58    ↪ getHorizontal()].item = item;
59        }
60    }
61
62    public void clearCell(int vertical, int horizontal) {

```

```

56         field[vertical][horizontal].item = new Empty(vertical,
57         ↪ horizontal);
58     }
59     public void clearCells(List<Coordinates> coordinatesList) {
60         for (Coordinates coordinates : coordinatesList) {
61             field[coordinates.getVertical()][coordinates.
62         ↪ getHorizontal()].item
63             = new Empty(coordinates.getVertical(),
64         ↪ coordinates.getHorizontal());
65         }
66     }
67     public Item getItem(int vertical, int horizontal) {
68         //TODO к примеру, здесь был бы get(), он, по идее, безопаснее
69
70         return field[vertical][horizontal].item; // todo мб return
71         ↪ new ...
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,
78         ↪ юезр поля не должен знать, что тут творится у( тебя же нет другого
79         ↪ метода получения размера поля), мне бы больше понравился простой
80         ↪ getSize()
81         return realSize;
82     }
83
84     public int getSize() {
85         return size;
86     }
87
88     /**
89     * Возвращает кратчайший путь в( координатах) из заданной позиции до
90     ↪ заданного ряда.
91     * Если нельзя пройти, вернется пустой стек.
92     * @param marker – координаты, из которых ищется путь
93     * @param rowNumber – номер ряда строки()
94     */
95     public Stack<Coordinates> getPathToRow(Coordinates marker, int
96     ↪ rowNumber) {
97
98         if (marker.getVertical() == rowNumber) {
99             Stack<Coordinates> coordinates = new Stack<>();
100             coordinates.add(getNeighbours(marker).get(0));
101             return coordinates;
102         }
103
104         class Vertex {
105             private Coordinates coordinates;
106             private Vertex from;
107
108             private Vertex(Coordinates coordinates, Vertex from) {
109                 this.coordinates = coordinates;
110                 this.from = from;
111             }
112         }

```

```

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() ==
124         ↪ rowNumber) {
125             Vertex vertex = queue.element();
126             while (vertex.from != null) {
127                 path.add(vertex.coordinates);
128                 vertex = vertex.from;
129             }
130             return path;
131         }
132         for (final Coordinates neighbour : getNeighbours(queue.
133         ↪ element().coordinates)) {
134             try {
135                 if (!used[neighbour.getVertical()][neighbour.
136         ↪ getHorizontal()] && // todo: шлифануть бы тут
137                 getItem((queue.element().coordinates.
138         ↪ getVertical() + neighbour.getVertical()) / 2,
139                 (queue.element().coordinates.
140         ↪ getHorizontal() + neighbour.getHorizontal()) / 2).getType()
141                 != ItemType.BARRIER &&
142                 !queue.contains(new Vertex(neighbour,
143         ↪ queue.element())))) {
144                     //Coordinates(neighbour.
145         ↪ getVertical(), neighbour.getHorizontal())) {
146                         queue.add(new Vertex(neighbour, queue.
147         ↪ element())); //neighbour
148                     } catch (ArrayIndexOutOfBoundsException e) { }
149                 }
150             }
151             used[queue.element().coordinates.getVertical()][queue.
152         ↪ element().coordinates.getHorizontal()] = true;
153             usedVertexes.add(queue.element());
154             queue.remove();
155         }
156         return path;
157     }
158
159     /**
160     * Возвращает кратчайший путь.
161     * @param init начальные координаты
162     * @param dest конечные координаты
163     * @return стек координат, по которым нужно пройти.
164     * Если пройти нельзя, вернется пустой стек.
165     */
166     //это метод для the Fox, 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() -
217 ↪ 2, coordinates.getHorizontal()));
217     neighbours.add(new Coordinates(coordinates.getVertical(),
218 ↪ coordinates.getHorizontal() - 2));
218     neighbours.add(new Coordinates(coordinates.getVertical() +
219 ↪ 2, coordinates.getHorizontal()));
219     neighbours.add(new Coordinates(coordinates.getVertical(),
220 ↪ coordinates.getHorizontal() + 2));
220
221     return neighbours;
222 }
223
224 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor;
2
3
4 import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.*;
5 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.*;
6
7 import java.util.*;
8
9
10 // todo разбить бы весь этот огромный класс. Мб сделать отдельный класс с
    ↪ логикой
11 // todo вообще, надо было без енама
12 public enum QuoridorPlayer {
13
14     TOP(0, 8, Owner.TOP, 16),
15     BOTTOM(16, 8, Owner.BOTTOM, 0);
16     // RIGHT(8, 16, Owner.RIGHT),
17     // LEFT(8, 0, Owner.LEFT);
18
19     private final int initialVertical;
20     private final int initialHorizontal;
21     private final Owner owner;
22     private final int destinationRow;
23
24     private Marker marker;
25     private boolean isBot = false;
26     private int barriersNumber = 10;
27     private List<Barrier> barriers = new ArrayList<Barrier>();
28     private QuoridorField field;
29     private boolean isActive = false;
30
31     QuoridorPlayer(int initialVertical, int initialHorizontal,
    ↪ Owner owner, int destinationRow) {
32         this.initialVertical = initialVertical;
33         this.initialHorizontal = initialHorizontal;
34         this.owner = owner;
35         this.destinationRow = destinationRow;
36     }
37
38     public void createPlayer(QuoridorField field, boolean isBot) {
    ↪ //TODO смущает, что чтобы создать игрока ему нужно
    ↪ передать поле
39         this.field = field;
40         this.isBot = isBot;
41         this.isActive = true;
42         barriersNumber = 10;
43         marker = new Marker(initialVertical, initialHorizontal,
    ↪ owner);
44         field.setItem(marker);
45     }
46
47     public Marker getMarker() { return marker; }
48
49     public int getBarriersNumber() { return barriersNumber; }
50
51     public int getDestinationRow() {
52         return destinationRow;
53     }
54
55     public List<Barrier> getBarriers() { return barriers; }
56
57     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.EMPTY) {
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.

```

```

110 ↪ getCoordinates(), TOP.destinationRow);
    Coordinates between = new Coordinates((topPath.peek().
111 ↪ getVertical() + TOP.marker.getCoordinates().getVertical()) /
    ↪ 2,
    (topPath.peek().getHorizontal() + TOP.marker.
112 ↪ getCoordinates().getHorizontal()) / 2);
    double rand = Math.random();
113     try {
114         if (rand < 0.5) {
115             makeMove(between.getVertical() % 2 == 0 ? between.
116 ↪ getVertical() - 1 : between.getVertical(),
                between.getHorizontal() % 2 == 0 ? between.
117 ↪ getHorizontal() - 1 : between.getHorizontal(),
                rand > 0.15 ? BarrierPosition.HORIZONTAL :
118 ↪ BarrierPosition.VERTICAL);
        } else {
119             makeMove(between.getVertical() % 2 == 0 ? between.
120 ↪ getVertical() + 1 : between.getVertical(),
                between.getHorizontal() % 2 == 0 ? between.
121 ↪ getHorizontal() + 1 : between.getHorizontal(),
                rand > 0.65 ? BarrierPosition.HORIZONTAL :
122 ↪ BarrierPosition.VERTICAL);
        }
123     } catch (Exception e) {
124         setBotMarker();
125     }
126 }
127
128
129 public void makeMove(int vertical, int horizontal) throws
130 ↪ FieldItemException {
131     checkPlace(vertical, horizontal);
132     setItem(vertical, horizontal);
133 }
134
135 public void makeMove(int vertical, int horizontal,
136 ↪ BarrierPosition position) throws FieldItemException,
137 ↪ NoBarriersException {
138     //TODO зачем называть метод makeMove, когда совершается
139 ↪ placeBarrier это( имя метода из Quoridor)
140     if (barriersNumber == 0) {
141         throw new NoBarriersException("you_have_no_barriers");
142 ↪ //TODO можно ещё передать у кого из игроков
143     }
144     checkPlace(vertical, horizontal, position);
145     setItem(vertical, horizontal, position);
146 }
147
148 private void checkPlace(int vertical, int horizontal) throws
149 ↪ FieldItemException {
150     try {
151         field.getItem(vertical, horizontal);
152     } catch (ArrayIndexOutOfBoundsException e) {
153         throw new FieldBoundsException("impossible_to_place_
154 ↪ marker_on_" + vertical + "_" + horizontal);
    }

```

```

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,
243 ↪ BarrierPosition position) throws FieldItemException {
244     //TODO я бы переименовал в checkPlaceForBarrier
245
246     if (vertical % 2 == 0 || horizontal % 2 == 0) {
247         throw new ImpossibleToSetItemException("impossible_to_
248 ↪ set_barrier_here");
249     }
250
251     if (position == BarrierPosition.VERTICAL) {
252 ↪ //todo что-то сделать
253         for (int i = vertical - Barrier.length + 1; i <=
254 ↪ vertical + Barrier.length - 1; i++) {
255             try {
256                 if (field.getItem(i, horizontal).getType() !=
257 ↪ ItemType.EMPTY) {
258                     throw new CellIsNotEmptyException("
259 ↪ impossible_to_place_barrier_here");
260                 }
261                 if (field.getColor(i, horizontal) == CellColor.
262 ↪ BLACK) {
263                     throw new ImpossibleToSetItemException("
264 ↪ impossible_to_set_barrier_on_black_cell");
265                 }
266             } catch (ArrayIndexOutOfBoundsException e) {
267                 throw new FieldBoundsException("impossible_to_
268 ↪ place_barrier_here");
269             }
270         }
271     } else if (position == BarrierPosition.HORIZONTAL) {
272         for (int i = horizontal - Barrier.length + 1; i <=
273 ↪ horizontal + Barrier.length - 1; i++) {
274             try {
275                 if (field.getItem(vertical, i).getType() !=
276 ↪ ItemType.EMPTY) {
277                     throw new CellIsNotEmptyException("
278 ↪ impossible_to_place_barrier_here");
279                 }
280                 if (field.getColor(vertical, i) == CellColor.
281 ↪ BLACK) {
282                     throw new ImpossibleToSetItemException("
283 ↪ impossible_to_set_barrier_on_black_cell");
284                 }
285             } catch (ArrayIndexOutOfBoundsException e) {
286                 throw new FieldBoundsException("impossible_to_
287 ↪ place_barrier_here");
288             }
289         }
290     }
291
292     Barrier probableBarrier = new Barrier(vertical, horizontal,
293 ↪ position);
294     field.setItem(probableBarrier);
295
296     if (TOP.isActive && field.getPathToRow(TOP.getMarker().
297 ↪ getCoordinates(), 16).empty()) {
298         field.clearCells(probableBarrier.getCoordinates());
299         throw new ImpossibleToSetItemException("you_can't_place
300 ↪ _barrier_here._Player_is_locked");
301     }

```

```

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 }

```



```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor;
2
3
4 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Empty;
5 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.ItemType;
6 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Marker;
7 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Owner;
8
9 import java.util.EmptyStackException;
10 import java.util.Stack;
11
12 public class Fox {
13
14     private Marker marker;
15     private QuoridorField field;
16     private Marker target;
17
18     public Fox(QuoridorField field) {
19
20         this.field = field;
21         int x, y;
22         do {
23             x = (int) (Math.random() * 10) % 9 * 2;
24             y = (int) (Math.random() * 10) % 9 * 2;
25         } while (field.getItem(x, y).getType() != ItemType.EMPTY);
26         marker = new Marker(x, y, Owner.FOX);
27         field.setItem(marker);
28
29         double rand = Math.random();
30         System.out.println("fox_rand: " + rand);
31         if (rand > 0.5) {
32             target = QuoridorPlayer.TOP.getMarker();
33         } else {
34             target = QuoridorPlayer.BOTTOM.getMarker();
35         }
36     }
37
38     public boolean makeMove() {
39
40         Coordinates c = getNextCoordinates();
41         // не проверяем потому, что проверяется в getNextCoordinates();
42         field.setItem(new Empty(marker.getCoordinates().getVertical
↪ ( ), marker.getCoordinates().getHorizontal()));
43         marker.moveTo(c.getVertical(), c.getHorizontal());
44         field.setItem(marker);
45
46         return c.equals(target.getCoordinates());
47     }
48
49     public Marker getMarker() {
50         return marker;
51     }
52
53     public Coordinates getTarget() {
54         return target.getCoordinates();
55     }
56
57     private Coordinates getNextCoordinates() {
58
59         Stack<Coordinates> path = field.getPath(marker.
↪ getCoordinates(), target.getCoordinates());
60         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 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor;
2
3
4 public class Coordinates {
5
6     private int vertical;
7     private int horizontal;
8
9     public static double pathBetween(Coordinates coordinates1,
10 ↪ Coordinates coordinates2) {
11         //TODO конечно бред, но можно выделить private метод,
12 ↪ возведения в квадрат, по крайней мере, понятность кода увеличится
13         return Math.sqrt((coordinates1.getVertical() - coordinates2
14 ↪ .getVertical()) *
15 ↪ (coordinates1.getVertical() - coordinates2
16 ↪ .getVertical()) +
17 ↪ (coordinates1.getHorizontal() -
18 ↪ coordinates2.getHorizontal()) *
19 ↪ (coordinates1.getHorizontal() -
20 ↪ coordinates2.getHorizontal()));
21     }
22
23     public int getVertical() {
24         return vertical;
25     }
26
27     public int getHorizontal() {
28         return horizontal;
29     }
30
31     public Coordinates(int vertical, int horizontal) {
32         this.horizontal = horizontal;
33         this.vertical = vertical;
34     }
35
36     @Override
37     public boolean equals(Object o) { //TODO однобуквенное имя
38 ↪ выглядит не очень
39         if (this == o) return true;
40         if (o == null || getClass() != o.getClass()) return false;
41
42         Coordinates that = (Coordinates) o;
43
44         if (vertical != that.vertical) return false; //TODO както-
45 ↪ криво, предлагаю объединить в одно return условие;
46         return horizontal == that.horizontal;
47     }
48
49     @Override
50     public int hashCode() {
51         int result = vertical;
52         result = 31 * result + horizontal;
53         return result;
54     }
55 }

```

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

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.items;
2
3
4 abstract public class Item {
5
6     protected ItemType type = ItemType.EMPTY;
7
8     protected Owner owner = Owner.NOBODY;
9
10    public Item() {}
11
12    public ItemType getType() { return type; }
13
14    public Owner getOwner() { return owner; }
15 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.items;
2
3
4 public enum ItemType {
5
6     MARKER,
7     BARRIER,
8     EMPTY
9 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.items;
2
3
4 import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
5
6 abstract public class OneCellItem extends Item {
7
8     protected Coordinates coordinates = new Coordinates(0, 0);
9
10    public OneCellItem(int vertical, int horizontal) {
11        coordinates = new Coordinates(vertical, horizontal);
12    }
13
14    public Coordinates getCoordinates() { //TODO возвращать
15        ↪ копию, хотя если её изменить класс Coordinates, то и так нормально
16        return coordinates;
17    }
18 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.items;
2
3
4 import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
5
6 import java.util.ArrayList;
7 import java.util.List;
8
9 abstract public class ManyCellsItem extends Item {
10
11    protected List<Coordinates> coordinates = new ArrayList<
12        ↪ Coordinates>();
13
14    public ManyCellsItem() {}
15 }

```

```

15 public List<Coordinates> getCoordinates() { //TODO по
    ↪ моему, не правильно отдавать поле возвращать( 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 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;
2
3
4 public class FieldBoundsException extends
    ↳ ArrayIndexOutOfBoundsException {
5     public FieldBoundsException(String s) {
6         super(s);
7     }
8 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;
2
3
4 public class FieldItemException extends Exception {    //TODO
    ↳ Исключение Пункт Области, возможно здесь есть смысл, но я не могу
    ↳ сказать, когда конкретно кидать это исключение
5
6     public FieldItemException(String s) {
7         super(s);
8     }
9
10 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;
2
3
4 public class CellIsNotEmptyException extends FieldItemException {
5
6     public CellIsNotEmptyException(String s) {
7         super(s);
8     }
9
10 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;
2
3
4 public class ImpossibleToSetItemException extends
    ↳ FieldItemException {    //TODO пожалуйста сделай так, чтобы
    ↳ исключение содержало больше информации о себе, нежели одну строчку
5
6     public ImpossibleToSetItemException(String s) {
7         super(s);
8     }
9
10 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;
2
3
4 public class NoBarriersException extends Exception {
5
6     public NoBarriersException(String s) {
7         super(s);
8     }
9
10 }

```



```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;
2
3
4 public class NoWinnerException extends Exception {           //
5     ↪ TODO я бы не стал делать исключение для рядовой ситуации
6     public NoWinnerException(String s) {
7         super(s);
8     }
9 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions;
2
3
4 public class TooLongDistanceException extends FieldItemException {
5
6     public TooLongDistanceException(String s) {
7         super(s);
8     }
9 }
10 }

```

6.2 Консольное приложение

```
1 package ru.spbstu.icc.kspt.zhuikov.quoridor.console;
2
3
4 import ru.spbstu.icc.kspt.zhuikov.quoridor.Quoridor;
5 import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Player;
6
7 import java.util.Scanner;
8
9 public class ConsoleGame {
10
11     private static Quoridor game = new Quoridor(2, false);
12     private static ConsoleDrawer drawer = new ConsoleDrawer(game);
13
14     public static void main(String[] args) {
15
16         ConsoleGame consoleGame = new ConsoleGame();
17         consoleGame.launch();
18
19     }
20
21     public void launch() {
22
23         Scanner in = new Scanner(System.in);
24         drawer.drawPlayerInformation(Player.TOP);
25         drawer.drawField();
26         drawer.drawPlayerInformation(Player.BOTTOM);
27         drawer.drawTurn();
28         while (!game.isEnd()) {
29
30             try {
31                 Command command = CommandReader.read(in.nextLine())
32                 ↪ ;
33                 switch (command.getCommandType()) {
34                     case BARRIER:
35                         game.placeBarrier(command.getCoordinates().
36                         ↪ getVertical(),
37                         ↪ command.getCoordinates().
38                         ↪ getHorizontal(),
39                         ↪ command.
40                         ↪ getBarrierPosition());
41                         break;
42                     case MARKER:
43                         game.moveMarker(command.getCoordinates().
44                         ↪ getVertical(),
45                         ↪ command.getCoordinates().
46                         ↪ getHorizontal());
47                         break;
48                     case HELP:
49                         drawer.drawHelp();
50                         break;
51                 }
52             } catch (Exception e) {
53                 System.out.println(e.getMessage());
54                 System.out.println();
55                 drawer.drawTurn();
56                 continue;
57             }
58
59             System.out.println();
60             drawer.drawPlayerInformation(Player.TOP);
61         }
62     }
63 }
```

```
55         drawer.drawField();
56         drawer.drawPlayerInformation(Player.BOTTOM);
57         drawer.drawTurn();
58     }
59
60     drawer.drawWinner();
61 }
62
63 }
```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.console;
2
3
4 import ru.spbstu.icc.kspt.zhuikov.quoridor.CellColor;
5 import ru.spbstu.icc.kspt.zhuikov.quoridor.Quoridor;
6 import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
    ↪ NoWinnerException;
7 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Owner;
8 import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Field;
9 import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Player;
10
11 public class ConsoleDrawer {
12
13     private Quoridor game = null;
14
15     public ConsoleDrawer(Quoridor game) {
16         this.game = game;
17     }
18
19     public void drawField() {
20
21         Field field = game.getField();
22         for (int i = 0; i < field.getRealSize(); i++) {
23             for (int j = 0; j < field.getRealSize(); j++) {
24                 switch (field.getCell(i, j).getType()) {
25                     case EMPTY:
26                         ↪ CellColor.BLACK) {
27                             if (field.getCell(i, j).getColor() ==
28                                 System.out.print("O");
29                             } else {
30                                 System.out.print("_");
31                             }
32                             break;
33                     case BARRIER:
34                         System.out.print("#");
35                         break;
36                     case MARKER:
37                         ↪ .BOTTOM) {
38                             if (field.getCell(i, j).getOwner() == Owner
39                                 System.out.print("B");
40                             } else if (field.getCell(i, j).getOwner()
41                                 ↪ == Owner.TOP) {
42                                     System.out.print("T");
43                                 }
44                             }
45                         }
46                     }
47                 System.out.println();
48             }
49         }
50
51     public void drawPlayerInformation(Player player) {
52
53         System.out.println(player.name() + "_" + game.
54         ↪ getPlayerInformation(player).getBarrierNumber());
55     }
56
57     public void drawTurn() {
58
59         System.out.print(game.getCurrentPlayer().name() + "_player'
60         ↪ s_turn:_");
61     }

```

```

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[_]\n":\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 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.console;
2
3
4 import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
5 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;
6
7 public class Command {
8
9     private CommandType commandType;
10    private Coordinates coordinates;
11    private BarrierPosition barrierPosition;
12
13    public Command(CommandType commandType) {
14        this.commandType = commandType;
15    }
16
17    public CommandType getCommandType() {
18        return commandType;
19    }
20
21    public Coordinates getCoordinates() {
22        return coordinates;
23    }
24
25    public BarrierPosition getBarrierPosition() {
26        return barrierPosition;
27    }
28
29    public void setCoordinates(Coordinates coordinates) {
30        this.coordinates = coordinates;
31    }
32
33    public void setBarrierPosition(BarrierPosition barrierPosition)
34    ↪ {
35        this.barrierPosition = barrierPosition;
36    }
37 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.console;
2
3
4 public enum CommandType {
5
6     BARRIER,
7     MARKER,
8     HELP,
9     // HIDE_HELP
10
11 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.console;
2
3
4 import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
5 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;
6
7 public class CommandReader {
8
9     public static Command read(String string) throws
10     ↪ IllegalArgumentException {
11
12         Command command;
13
14         if (string.toLowerCase().matches("\\s*marker\\s+-?\\d+\\s
15     ↪ +-?\\d+\\s*")) {
16             command = new Command(CommandType.MARKER);
17             String[] stringArray = string.split("\\s+");
18             command.setCoordinates(new Coordinates(Integer.parseInt
19     ↪ (stringArray[1]),
20                                     Integer.parseInt
21     ↪ (stringArray[2])));
22             return command;
23         }
24
25         if (string.toLowerCase().matches("\\s*barrier\\s+-?\\d+\\s
26     ↪ +-?\\d+\\s+(horizontal|vertical)\\s*")) {
27             command = new Command(CommandType.BARRIER);
28             String[] stringArray = string.split("\\s+");
29             command.setCoordinates(new Coordinates(Integer.parseInt
30     ↪ (stringArray[1]),
31                                     Integer.parseInt
32     ↪ (stringArray[2])));
33             if (stringArray[3].equals("vertical")) {
34                 command.setBarrierPosition(BarrierPosition.VERTICAL
35     ↪ );
36             } else {
37                 command.setBarrierPosition(BarrierPosition.
38     ↪ HORIZONTAL);
39             }
40             return command;
41         }
42
43         if (string.toLowerCase().matches("\\s*help\\s*")) {
44             return new Command(CommandType.HELP);
45         }
46
47         // if (string.toLowerCase().matches("\\s*hide\\s+help\\s*"))
48     ↪ {
49             // return new Command(CommandType.HIDE_HELP);
50             // }
51
52         throw new IllegalArgumentException("unknown_command");
53     }
54 }

```

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    public static void main(String[] args) {
17        SwingUtilities.invokeLater(new Runnable() {
18            public void run() {
19                new MainFrame("Quoridor");
20            }
21        });
22    }
23 }
24 }
```



```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.gui;
2
3
4 import javax.swing.*;
5 import java.awt.*;
6 import java.awt.event.*;
7
8 public class MenuPanel extends JPanel {
9
10     private JButton startOnePlayer = new JButton("One_Player");
11     private JButton startTwoPlayer = new JButton("Two_Players");
12     private JButton settingsButton = new JButton("Settings");
13     private JButton exitButton = new JButton("Exit");
14     private JFrame frame;
15     private final Image bg = new ImageIcon("pictures/gamePics/
    ↪ background_menu1.jpg").getImage();
16
17     MenuPanel(MainFrame frame) {
18
19         this.frame = frame;
20         setLayout(null);
21
22         startOnePlayer.setLocation(225, 220);
23         startOnePlayer.setSize(200, 30);
24         startOnePlayer.addMouseListener(new StartListener(true));
25
26         startTwoPlayer.setLocation(225, 270);
27         startTwoPlayer.setSize(200, 30);
28         startTwoPlayer.addMouseListener(new StartListener(false));
29
30         settingsButton.setLocation(225, 320);
31         settingsButton.setSize(200, 30);
32         settingsButton.addMouseListener(new SettingsListener());
33
34
35         exitButton.setLocation(225, 370);
36         exitButton.setSize(200, 30);
37         exitButton.addActionListener(new ExitListener());
38
39         add(startOnePlayer);
40         add(startTwoPlayer);
41         add(settingsButton);
42         add(exitButton);
43     }
44
45     @Override
46     protected void paintComponent(Graphics g) {
47
48         super.paintComponent(g);
49         g.drawImage(bg, 0, 0, null);
50     }
51
52     private class StartListener implements MouseListener {
53
54         private boolean bots = false;
55         StartListener(boolean bots) {
56             this.bots = bots;
57         }
58
59         @Override
60         public void mouseClicked(MouseEvent e) {
61             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 }

```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.gui;
2
3
4 import ru.spbstu.icc.kspt.zhuikov.quoridor.Quoridor;
5
6 import javax.swing.*;
7 import java.awt.*;
8 import java.awt.event.MouseEvent;
9 import java.awt.event.MouseListener;
10
11 public class SettingsPanel extends JPanel {
12
13     private final Image bg = new ImageIcon("pictures/gamePics/
    ↪ background_menu1.jpg").getImage();
14     private MainFrame frame;
15     private JButton backButton = new JButton("Back");
16     private JSlider stepSlider = new JSlider(0, 40, Quoridor.
    ↪ getFoxTime());
17     private JSlider frequencySlider = new JSlider(1, 20, Quoridor.
    ↪ getFoxFrequency());
18     private JLabel stepLabel = new JLabel("");
19     private JLabel frequencyLabel = new JLabel("");
20
21     SettingsPanel(MainFrame frame) {
22
23         this.frame = frame;
24         setLayout(null);
25
26         JLabel changeFoxTime = new JLabel("Fox_spawn_time_(in_steps
    ↪ _from_beginning):_");
27         changeFoxTime.setLocation(130, 210);
28         changeFoxTime.setSize(400, 20);
29         changeFoxTime.setFont(new Font("Arial", Font.BOLD + Font.
    ↪ ITALIC, 15));
30
31         stepSlider.setSize(300, 45);
32         stepSlider.setFont(new Font("Arial", Font.BOLD + Font.ITALIC
    ↪ , 14));
33         stepSlider.setBackground(new Color(210, 230, 250));
34         stepSlider.setLocation(110, 240);
35         stepSlider.setMajorTickSpacing(5);
36         stepSlider.setPaintLabels(true);
37         stepSlider.setPaintTicks(true);
38         stepSlider.addChangeListener(e -> {
39             stepLabel.setText("" + stepSlider.getValue());
40             Quoridor.setFoxTime(stepSlider.getValue());
41         });
42
43         stepLabel = new JLabel("" + stepSlider.getValue());
44         stepLabel.setSize(140, 45);
45         stepLabel.setLocation(480, 242);
46         stepLabel.setFont(new Font("Arial", Font.BOLD + Font.ITALIC
    ↪ , 16));
47
48         JLabel changeFoxTurn = new JLabel("Fox_move_frequency_(once
    ↪ _in_a_set_steps):_");
49         changeFoxTurn.setLocation(130, 300);
50         changeFoxTurn.setSize(400, 20);
51         changeFoxTurn.setFont(new Font("Arial", Font.BOLD + Font.
    ↪ ITALIC, 15));
52
53         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 | }
```

```

1 package ru.spbstu.icc.kspt.zhuikov.quoridor.gui;
2
3 import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
4 import ru.spbstu.icc.kspt.zhuikov.quoridor.Quoridor;
5 import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
    ↪ FieldItemException;
6 import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
    ↪ NoBarriersException;
7 import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
    ↪ NoWinnerException;
8 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;
9 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.ItemType;
10 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Owner;
11 import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Cell;
12 import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Field;
13 import ru.spbstu.icc.kspt.zhuikov.quoridor.returningClasses.Player;
14
15 import javax.swing.*;
16 import javax.swing.border.LineBorder;
17 import java.awt.*;
18 import java.awt.event.*;
19
20 class GamePanel extends JPanel {
21
22     private final Color bottomColor = Color.red;
23     private final Color topColor = Color.blue;
24
25     private MainFrame frame;
26     private Quoridor game;
27     private FieldPanel fieldPanel;
28     private BarrierPanel barrierPanel;
29     private JLabel statusLabel;
30     private FoxPanel foxPanel;
31     private Image bg = new ImageIcon("pictures/gamePics/
    ↪ background_game.jpg").getImage();
32
33     GamePanel(MainFrame frame, boolean bots) {
34
35         this.frame = frame;
36         game = new Quoridor(2, bots);
37
38         statusLabel = new JLabel("fff");
39         statusLabel.setSize(470, 25);
40         statusLabel.setLocation(45, 395);
41         statusLabel.setBorder(new LineBorder(Color.BLACK, 2));
42         statusLabel.setFont(new Font("Arial", Font.ITALIC + Font.
    ↪ BOLD, 17));
43
44         updateStatusLabel();
45         add(statusLabel);
46
47         JButton menuButton = new JButton("Menu");
48         menuButton.setLocation(435, 20);
49         menuButton.setSize(140, 20);
50         menuButton.addActionListener(new MenuListener());
51         add(menuButton);
52
53         fieldPanel = new FieldPanel();
54         barrierPanel = new BarrierPanel();
55         foxPanel = new FoxPanel();
56
57         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
126 private void unpickBarrier(int x, int y) {
127     Coordinates coordinates = roundToOdd(x, y);
128     try {
129         game.placeBarrier(coordinates.getVertical(),
130             ↪ coordinates.getHorizontal(), barrierPosition);
131         pickedBarrier = false;
132         updateStatusLabel();
133         barrierPanel.updateText();
134     } catch (FieldItemException | NoBarriersException e) {
135         statusLabel.setText(e.getMessage());
136         pickedBarrier = false;
137         Timer timer = new Timer(1500, new ActionListener()
138             ↪ {
139                 @Override
140                 public void actionPerformed(ActionEvent e) {
141                     updateStatusLabel();
142                 }
143             });
144         timer.setRepeats(false);
145         timer.start();
146         // barrierPanel.updateText();
147     }
148 }
149
150 private Coordinates roundToOdd(int x, int y) {
151     Coordinates initCoordinates = convertCoordinates(x, y);
152     int newX = x;
153     int newY = y;
154     if (initCoordinates.getVertical() % 2 == 0) {
155         ↪ int leftBound = (initCoordinates.getVertical() /
156             ↪ 2) * (cellSize + spaceSize) - spaceSize / 2; // в пикселах
157         ↪ int rightBound = (initCoordinates.getVertical() / 2
158             ↪ + 1) * (cellSize + spaceSize) - spaceSize / 2;
159         ↪ if (Math.abs(y - leftBound) < Math.abs(y -
160             ↪ rightBound)) {
161             newY = leftBound;
162         } else {
163             newY = rightBound;
164         }
165     }
166     if (initCoordinates.getHorizontal() % 2 == 0) {
167         ↪ int leftBound = (initCoordinates.getHorizontal() /
168             ↪ 2) * (cellSize + spaceSize) - spaceSize / 2; // в пикселах
169         ↪ int rightBound = (initCoordinates.getHorizontal() /
170             ↪ 2 + 1) * (cellSize + spaceSize) - spaceSize / 2;
171         ↪ if (Math.abs(x - leftBound) < Math.abs(x -
172             ↪ rightBound)) {
173             newX = leftBound;
174         } else {
175             newX = rightBound;
176         }
177     }
178 }

```



```

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

```

```

219     ↪ - 3 },
                                new int [] {i * (cellSize +
220     ↪ spaceSize) / 2 + 3, i * (cellSize + spaceSize) / 2 +
221     ↪ cellSize - 3, i * (cellSize + spaceSize) / 2 + 3}, 3);
                                }
222     ↪ break;
                                }
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 +
227     ↪ spaceSize) / 2 - spaceSize,
                                (i + 1) * (cellSize +
228     ↪ spaceSize) / 2 - spaceSize, spaceSize, spaceSize);
                                if (i % 2 == 1 && j % 2 == 0)
229     ↪         g.fillRect(j * (cellSize +
230     ↪ spaceSize) / 2,
                                (i + 1) * (cellSize +
231     ↪ spaceSize) / 2 - spaceSize, cellSize, spaceSize);
                                if (i % 2 == 0 && j % 2 == 1)
232     ↪         g.fillRect((j + 1) * (cellSize +
233     ↪ spaceSize) / 2 - spaceSize,
                                i * (cellSize + spaceSize)
234     ↪ / 2, spaceSize, cellSize);
                                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()
247     ↪ * (cellSize + spaceSize) / 2,
                                pickedMarkerCoordinates.getVertical() * (
248     ↪ cellSize + spaceSize) / 2, cellSize, cellSize);
                                for (Coordinates c : game.getPossibleMoves()) {
249     ↪         g.setColor(Color.green);
250     ↪         g.fillRect(c.getHorizontal() * (cellSize +
251     ↪ spaceSize) / 2,
                                c.getVertical() * (cellSize + spaceSize
252     ↪ ) / 2, cellSize, cellSize);
                                }
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("
262     ↪ BOTTOM")) {
                                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 private class BarrierPanel extends JPanel {
356
357     private JLabel bottomBarriersNumber = new JLabel();
358     private JLabel topBarriersNumber = new JLabel();
359     private JButton barrierButton = new JButton("Place_barrier"
↪ );
360     private JButton cancelButton = new JButton("Cancel");
361     private BarrierListener barrierListener;
362     private CancelBarrierListener cancelBarrierListener;
363     private JRadioButton horizontal = new JRadioButton("
↪ Horizontal", true);
364     private JRadioButton vertical = new JRadioButton("Vertical"
↪ , false);
365
366     BarrierPanel() {
367
368         Color color = new Color(190, 140, 140);
369         setLayout(null);
370
371         setBorder(new LineBorder(Color.BLACK, 2));
372         setLocation(410, 180);
373         setSize(190, 200);
374         setBackground(color);
375         setVisible(true);
376     }

```

```

377         bottomBarriersNumber.setLocation(15, 10);
378         bottomBarriersNumber.setSize(185, 20);
379         bottomBarriersNumber.setFont(new Font("Arial", Font.
380 ↪ ITALIC + Font.BOLD, 14));
381         topBarriersNumber.setLocation(10, 30);
382         topBarriersNumber.setSize(185, 20);
383 ↪         topBarriersNumber.setFont(new Font("Arial", Font.BOLD +
384 ↪ Font.ITALIC, 14));
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     void removeListeners() {
417         barrierButton.removeMouseListener(barrierListener);
418         cancelButton.removeMouseListener(cancelBarrierListener)
419 ↪ ;
420     }
421
422     void updateText() {
423         bottomBarriersNumber.setText("Red_has_" + game.
424 ↪ getPlayerInformation(Player.BOTTOM).getBarrierNumber() + "_"
425 ↪ barriers");
426         topBarriersNumber.setText("Blue_has_" + game.
427 ↪ getPlayerInformation(Player.TOP).getBarrierNumber() + "_"
428 ↪ barriers");
429     }
430
431     private class BarrierListener implements MouseListener {
432         @Override
433         public void mouseClicked(MouseEvent e) {
434             fieldPanel.pickBarrier(horizontal.isSelected() ?

```

```

432 ↪ BarrierPosition.HORIZONTAL : BarrierPosition.VERTICAL);
433     }
434     @Override
435     public void mousePressed(MouseEvent e) {
436     }
437
438     @Override
439     public void mouseReleased(MouseEvent e) {
440     }
441
442     @Override
443     public void mouseEntered(MouseEvent e) {
444     }
445
446     @Override
447     public void mouseExited(MouseEvent e) {
448     }
449
450     private class CancelBarrierListener implements
451     ↪ MouseListener {
452         @Override
453         public void mouseClicked(MouseEvent e) {
454             fieldPanel.pickedBarrier = false;
455             updateStatusLabel();
456         }
457
458         @Override
459         public void mousePressed(MouseEvent e) {
460         }
461
462         @Override
463         public void mouseReleased(MouseEvent e) {
464         }
465
466         @Override
467         public void mouseEntered(MouseEvent e) {
468         }
469
470         @Override
471         public void mouseExited(MouseEvent e) {
472         }
473     }
474 }
475
476 class FoxPanel extends JPanel {
477     private JLabel foxAppearingLabel;
478     private JLabel foxFrequencyLabel;
479     FoxPanel() {
480     }
481
482
483
484
485
486
487
488
489
490
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.spbstu.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,
18         ↪ NoBarriersException {
19
20         QuoridorField field = new QuoridorField(9);
21         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
22         player.createPlayer(field, false);
23
24         player.makeMove(5, 3, BarrierPosition.VERTICAL);
25
26         assertEquals(ItemType.BARRIER, field.getItem(5, 3).getType()
27             ↪ ());
28         assertEquals(ItemType.BARRIER, field.getItem(4, 3).getType()
29             ↪ ());
30         assertEquals(ItemType.BARRIER, field.getItem(6, 3).getType()
31             ↪ ());
32
33         player.makeMove(13, 11, BarrierPosition.HORIZONTAL);
34
35         assertEquals(ItemType.BARRIER, field.getItem(13, 10).
36             ↪ getType());
37         assertEquals(ItemType.BARRIER, field.getItem(13, 11).
38             ↪ getType());
39         assertEquals(ItemType.BARRIER, field.getItem(13, 12).
40             ↪ getType());
41     }
42
43     @Test(expected = ImpossibleToSetItemException.class)
44     public void testBlackCellSet() throws FieldItemException,
45         ↪ NoBarriersException {
46
47         QuoridorField field = new QuoridorField(9);
48         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
49         player.createPlayer(field, false);
50         player.makeMove(2, 8, BarrierPosition.HORIZONTAL);
51     }
52
53     @Test(expected = ImpossibleToSetItemException.class)
54     public void testSetBetweenBlackCells() throws
55         ↪ FieldItemException, NoBarriersException {
56
57         QuoridorField field = new QuoridorField(9);
58         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
59         player.createPlayer(field, false);
```



```

52         player.makeMove(7, 12, BarrierPosition.VERTICAL);
53     }
54
55     // @Test(expected = FieldBoundsException.class)
56     // public void testWrongCoordinates() throws FieldItemException,
57     // ↪ NoBarriersException {
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,
66     ↪ NoBarriersException {
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,
77     ↪ NoBarriersException {
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
84             ↪ barriers
85         }
86         player.makeMove(5, 1, BarrierPosition.VERTICAL);
87         player.makeMove(5, 3, BarrierPosition.VERTICAL);
88
89         player.makeMove(5, 5, BarrierPosition.VERTICAL);
90     }
91
92     @Test (expected = ImpossibleToSetItemException.class)
93     public void testPlayerBlock() throws FieldItemException,
94     ↪ NoBarriersException {
95         QuoridorField field = new QuoridorField(9);
96         QuoridorPlayer bottom = QuoridorPlayer.BOTTOM;
97         QuoridorPlayer top = QuoridorPlayer.TOP;
98         bottom.createPlayer(field, false);
99         top.createPlayer(field, false);
100
101         field.setItem(new Barrier(1, 7, BarrierPosition.VERTICAL));
102         field.setItem(new Barrier(1, 9, BarrierPosition.VERTICAL));
103
104         bottom.makeMove(3, 8, BarrierPosition.HORIZONTAL);
105     }

```

```

1 package ru.spbstu.icc.kspt.Zhuikov.courseWork;
2
3
4 import org.junit.Test;
5 import ru.spbstu.icc.kspt.zhuikov.quoridor.CellColor;
6 import ru.spbstu.icc.kspt.zhuikov.quoridor.Coordinates;
7 import ru.spbstu.icc.kspt.zhuikov.quoridor.QuoridorField;
8 import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.
    ↪ FieldItemException;
9 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Barrier;
10 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;
11
12 import static org.junit.Assert.assertEquals;
13
14 public class FieldTest {
15
16     @Test
17     public void testInitialization() {
18
19         QuoridorField field = new QuoridorField(9);
20
21         assertEquals(CellColor.BLACK, field.getColor(0, 0));
22         assertEquals(CellColor.WHITE, field.getColor(0, 1));
23         assertEquals(CellColor.WHITE, field.getColor(1, 1));
24         assertEquals(CellColor.BLACK, field.getColor(2, 2));
25     }
26
27     @Test
28     public void testPathBetweenCells1() { // пустое поле
29
30         QuoridorField field = new QuoridorField(9);
31         assertEquals(false, field.getPathToRow(new Coordinates(16,
    ↪ 8), 0).empty());
32         assertEquals(false, field.getPathToRow(new Coordinates(0,
    ↪ 8), 10).empty());
33     }
34
35     @Test
36     public void testPathBetweenCells2() throws FieldItemException {
    ↪ // перегородка "" по вертикали
37
38         QuoridorField field = new QuoridorField(9);
39
40         for (int i = 1; i <= 13; i+=4) {
41             field.setItem(new Barrier(i, 7, BarrierPosition.
    ↪ VERTICAL));
42         }
43         field.setItem(new Barrier(15, 7, BarrierPosition.HORIZONTAL
    ↪ ));
44         field.setItem(new Barrier(15, 9, BarrierPosition.VERTICAL))
    ↪ ;
45
46         assertEquals(false, field.getPathToRow(new Coordinates(16,
    ↪ 8), 0).empty());
47         assertEquals(false, field.getPathToRow(new Coordinates(0,
    ↪ 16), 0).empty());
48         assertEquals(false, field.getPathToRow(new Coordinates(0,
    ↪ 16), 16).empty());
49     }
50
51     @Test
52     public void testPathBetweenCells3() throws FieldItemException {

```

```

53 ↪ // перегородка "" по горизонтали
54     QuoridorField field = new QuoridorField(9);
55     for (int i = 1; i <= 13; i+=4) {
56         field.setItem(new Barrier(7, i, BarrierPosition.
57 ↪ HORIZONTAL));
58     }
59     field.setItem(new Barrier(7, 15, BarrierPosition.VERTICAL))
60 ↪ ;
61     field.setItem(new Barrier(5, 15, BarrierPosition.HORIZONTAL
62 ↪ ));
63     assertEquals(false, field.getPathToRow(new Coordinates(0,
64 ↪ 4), 2).empty());
65     assertEquals(true, field.getPathToRow(new Coordinates(0, 4)
66 ↪ , 10).empty());
67     assertEquals(true, field.getPathToRow(new Coordinates(16,
68 ↪ 6), 0).empty());
69 }
70
71 @Test
72 public void testPathBetweenCells4() throws FieldItemException {
73 ↪ // закрытая фишка
74
75     QuoridorField field = new QuoridorField(9);
76     field.setItem(new Barrier(15, 7, BarrierPosition.VERTICAL))
77 ↪ ;
78     field.setItem(new Barrier(15, 9, BarrierPosition.VERTICAL))
79 ↪ ;
80     field.setItem(new Barrier(13, 8, BarrierPosition.HORIZONTAL
81 ↪ ));
82
83     assertEquals(true, field.getPathToRow(new Coordinates(16,
84 ↪ 8), 6).empty());
85     assertEquals(false, field.getPathToRow(new Coordinates(16,
86 ↪ 8), 16).empty());
87     assertEquals(false, field.getPathToRow(new Coordinates(0,
88 ↪ 8), 16).empty());
89 }
90 }

```

```

1 package ru.spbstu.icc.kspt.Zhuikov.courseWork;
2
3
4 import org.junit.Test;
5
6 import ru.spbstu.icc.kspt.zhuikov.quoridor.QuoridorField;
7 import ru.spbstu.icc.kspt.zhuikov.quoridor.QuoridorPlayer;
8 import ru.spbstu.icc.kspt.zhuikov.quoridor.exceptions.*;
9 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Barrier;
10 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.BarrierPosition;
11 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.ItemType;
12 import ru.spbstu.icc.kspt.zhuikov.quoridor.items.Marker;
13
14 import static org.junit.Assert.assertEquals;
15
16 public class MarkerTest {
17
18     @Test
19     public void testInitialCoordinates() {
20
21         QuoridorField field = new QuoridorField(9);
22         QuoridorPlayer player1 = QuoridorPlayer.TOP;
23         QuoridorPlayer player2 = QuoridorPlayer.BOTTOM;
24         player1.createPlayer(field, false);
25         player2.createPlayer(field, false);
26
27         assertEquals(ItemType.MARKER, field.getItem(0, 8).getType()
28 ↪ );
29         assertEquals(ItemType.MARKER, field.getItem(16, 8).getType
30 ↪ ());
31     }
32
33     @Test
34     public void testMoving() throws FieldItemException {
35
36         QuoridorField field = new QuoridorField(9);
37         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
38         player.createPlayer(field, false);
39
40         player.makeMove(14, 8);
41
42         assertEquals(ItemType.MARKER, field.getItem(14, 8).getType
43 ↪ ());
44     }
45
46     @Test (expected = FieldBoundsException.class)
47     public void testMovingOutOfBounds() throws FieldItemException {
48
49         QuoridorField field = new QuoridorField(9);
50         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
51         player.createPlayer(field, false);
52
53         player.makeMove(18, 8);
54     }
55
56     @Test (expected = TooLongDistanceException.class)
57     public void testFarMoving() throws FieldItemException {
58
59         QuoridorField field = new QuoridorField(9);
60         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
79         QuoridorField field = new QuoridorField(9);
80         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
81         player.createPlayer(field, false);
82
83         player.makeMove(16, 8);
84     }
85
86     @Test (expected = ImpossibleToSetItemException.class)
87     public void testJumpOverBarrier() throws FieldItemException {
88
89         QuoridorField field = new QuoridorField(9);
90         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
91         player.createPlayer(field, false);
92         Barrier barrier = new Barrier(15, 8, BarrierPosition.
93     ↪ HORIZONTAL);
94         field.setItem(barrier);
95
96         player.makeMove(14, 8);
97     }
98
99     @Test
100     public void testJumpOverMarker_Forward() throws
101     ↪ FieldItemException {
102
103         QuoridorField field = new QuoridorField(9);
104         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
105         player.createPlayer(field, false);
106         field.setItem(new Marker(14, 8));
107
108         player.makeMove(12, 8);
109         assertEquals(ItemType.MARKER, field.getItem(12, 8).getType
110     ↪ ());
111     }
112
113     @Test
114     public void testJumpOverMarker_Diagonal() throws
115     ↪ FieldItemException {
116
117         QuoridorField field = new QuoridorField(9);
118         QuoridorPlayer player = QuoridorPlayer.BOTTOM;
119         player.createPlayer(field, false);
120         field.setItem(new Marker(14, 8));

```

```

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 }

```