Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Волгоградский государственный технический университет»

Факультет электроники и вычислительной техники
Кафедра «Программное обеспечение автоматизированных систем»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе

по дисциплине «Объектно-ориентированный анализ и программирование»
на тему: «Проектирование программы с использованием объектно-ориентированного подхода»
(индивидуальное задание – вариант №11_01)

Студент: Козарез М.В.
Группа: ПрИн-366

Работа зачтена с оценкой _____ «__» __июня_ 2023 г.

Руководитель проекта, нормоконтроллер _____ Литовкин Д.В.

Волгоград 2023 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Волгоградский государственный технический университет»

Факультет электроники и вычислительной техники
Направление 09.03.04 «Программная инженерия»
Кафедра «Программное обеспечение автоматизированных систем»

Дисциплина «Объектно-ориентированный анализ и программирование»

Утверждаю
Зав. кафедрой _____ Орлова Ю.А.

ЗАДАНИЕ
на курсовую работу

Студент: Козарез М.В.
Группа: ПрИн-366

1. Тема: «Проектирование-программы с использованием объектно-ориентированного подхода» (индивидуальное задание – вариант №11_01)
Утверждена приказом от «__» января 2023г. № 101-ст

2. Срок представления работы к защите « 04 » июня 2023 г.

3. Содержание пояснительной записки:
формулировка задания, требования к программе, структура программы, типовые процессы в программе, человеко-машинное взаимодействие, код программы и модульных тестов

4. Перечень графического материала:
_____

5. Дата выдачи задания «13» февраля 2023 г.

Руководитель проекта: _____ Литовкин Д.В.

Задание принял к исполнению: _____ Козарез М.В.

«13» февраля 2023 г.

Содержание

1 Формулировка задания

Правила игры «Балда»:

- игра происходит на квадратном поле N x N, разделенном на ячейки;
- в центре поля, по горизонтали, расположено случайное слово длиной, равной длине поля;
- игроки по очереди ставят любую букву русского алфавита в пустые клетки, смежные с занятыми посредством визуального алфавита;
- после установки буквы, походивший игрок должен задать слово, путём нажатия на клетки, так, чтобы каждая выбранная клетка была смежной с уже выбранной. Одной из выбранных клеток должна быть та, в которую только что была поставлена буква
- игрок может пропустить ход
- игра заканчивается, если оба игрока пропустят ход
- за каждый ход игрок получает количество очков, равное длине выбранного слова
- игра заканчивается после того, как последняя клетка была занята
- побеждает игрок, набравший наибольшее количество очков

Дополнительные требования:

- необходимо предусмотреть в программе **точки расширения**, используя которые можно реализовать вариативную часть программы (в дополнение к базовой функциональности).

Вариативность:

- предусмотреть различный уровень сложности игры, который определяет специфическое поведение игрового поля и алфавита. В процессе игры поле может блокировать доступность ячеек, а алфавит - букв.

Реализовать:

- размер игрового поля можно изменять;
- игра для двух игроков людей;
- имеется возможность добавить в словарь новое слово, набранное на поле, во время игры

2 Нефункциональные требования

1. Программа должна быть реализована на языке Java SE 17 с использованием стандартных библиотек, в том числе, библиотеки Swing.
2. Форматирование исходного кода программы должно соответствовать Java Code Conventions, September 12, 1997.

3 Первая итерация разработки

3.1 Формулировка упрощенного варианта задания

Правила игры «Балда»:

- Инициация нового сеанса игры или инициация переигровки;
- Создание поля размером NxN клеток и отображение его;
- Задание случайного слова в центре по горизонтали;
- Определение игрока, который должен ходить, и предоставление ему хода;
- Проверка выбранного игроком слова в словаре;
- Возможность добавить набранное на поле слово в словарь по ходу игры;
- Определение того факта, что на поле нет незанятых клеток;
- Определение игрока-победителя (на основе набранных игроками очков), и выдача сообщения об этом;
- Установка игроком буквы в выбранную незанятую клетку, имеющую рядом занятую клетку;

3.2 Функциональные требования (сценарии)

1) Главный успешный сценарий — **Победил один из игроков**

1. Один из двух игроков инициирует новую сессию игры
2. Создается singleton класс словарь
3. Игрок, инициализовавший новую игру, устанавливает размер поля
4. Модель создает поле размером NxN клеток
5. Игровая панель создает визуальный алфавит, список составленных игроками слов и активные кнопки, и отображает все перечисленное, включая созданное моделью поле
6. Модель устанавливает в центр поля случайное слово, равное длине поля
7. Модель назначает первого игрока текущим
8. **Делать**
    - 8.1. Текущий игрок выбирает букву посредством визуального алфавита
    - 8.2. Текущий игрок выбирает свободную клетку, соседнюю занятой
    - 8.3. Игрок выбирает последовательность рядом стоящих букв, включая только что поставленную им, чтобы составить слово

8.4. Модель проверяет наличие заданного игроком слова в словаре, **если** слово найдено - добавляет слово в таблицу слов, составленных игроком, и увеличивает счет игрока, составившего его на значение равное длине слова, **иначе** предлагает выбор

8.4.1. Добавить слово в словарь:

8.4.1.1. Игра продолжается, начиная с пункта 8.5

8.4.2. Отменить добавления слова

8.4.2.1. Все действия текущего игрока за этот ход обнуляются и ход начинается заново

8.5. Обработчик счёта отображает обновленный счет игрока

8.6. Словарь игроков отображает, только что добавленное игроком слово

8.7. Модель считает, что игра не завершена, **если** на поле есть свободные клетки; **иначе** она считает, что игра завершена

8.8. Система передает ход другому игроку, **если** игра не завершена

Пока игра не завершена

9. Панель определяет победителем игрока, набравшего больше очков, и выводит сообщение

1.1) Альтернативный сценарий — **Ничья**

Сценарий начинает после п.7 сценария 1)

1. Делать

1.1. Текущий игрок выбирает букву посредством визуального алфавита

1.2. Текущий игрок выбирает свободную клетку, соседнюю занятой

1.3. Игрок выбирает последовательность рядом стоящих букв, включая только что поставленную им, чтобы составить слово

1.4. Модель проверяет наличие заданного игроком слова в словаре, **если** слово найдено - добавляет слово в таблицу слов, составленных игроком, и увеличивает счет игрока, составившего его на значение равное длине слова, **иначе** предлагает выбор

1.4.1. Добавить слово в словарь:

1.4.1.1. Игра продолжается, начиная с пункта 8.5

1.4.2. Отменить добавления слова

1.4.2.1. Все действия текущего игрока за этот ход обнуляются и ход начинается заново

1.5. Обработчик счёта отображает обновленный счет игрока

1.6. Словарь игроков отображает, только что добавленное игроком слово

1.7. Модель считает, что игра не завершена, **если** на поле есть свободные клетки; **иначе** она считает, что игра завершена

1.8. Система передает ход другому игроку, **если** игра не завершена

Пока игра не завершена (имеются незанятые клетки)

2. Панель определяет, что счёт равный и выводит сообщение

1.2) Альтернативный сценарий — **Победил один из игроков (оба игрока сдались)**

Сценарий начинает в любой итерации п.8 сценария 1)

1. Текущий игрок пропускает ход
2. Система передает право хода другому игроку
3. Текущий игрок пропускает ход
4. Панель определяет победителем игрока, набравшего наибольшее количество очков, и выводит сообщение

1.3) Альтернативный сценарий — **Ничья (оба игрока сдались)**

Сценарий начинает в любой итерации п.8 сценария 1)

1. Текущий игрок пропускает ход
2. Система передает право хода другому игроку
3. Текущий игрок пропускает ход
4. Панель определяет, что счёт равный, и выводит сообщение

1.4) Альтернативный сценарий главного успешного сценария — **Досрочное завершение игры (всей программы)**

- Сценарий начинается в любой точке сценария 1), **когда** любой из двух игроков инициирует завершение игры
- Система завершает свою работу без каких-либо запросов и предупреждений

2) Успешный сценарий - **Текущий игрок последовательно выбирает занятые клетки соседние последней выбранной для составления слова**

Делать

1. Игрок указывает занятую клетку

2. Поле проверяет, содержит ли клетка поставленную игроком букву, **если** выбранная клетка содержит только что поставленную букву, **то** сделать активной кнопку завершения хода, **иначе** оно считает, что ход еще не может быть завершен

3. Поле показывает список доступных для выбора клеток для составления слова

Пока игрок не подтвердит свой выбор клеток

- Сценарий продолжается после п. 8.3 сценария 1)

2.1) Альтернативный сценарий - **Текущий игрок последовательно выбирает незанятые клетки для составления слова, не выбрав ту, в которую поставил букву**

Делать

1. Сценарий начинает после п.2 сценария 2)

2. В любой момент хода, после установки буквы на поле, у игрока будут доступны две опции – отменить ход и пропустить ход. Так, игрок будет вынужден сделать выбрать один из двух вариантов

Пока игрок не подтвердит свой выбор клеток

- Сценарий начинается заново, с пункта 8 сценария 1)

3.3 Словарь предметной области

Модель – система знает о поле. Модель управляет игровым циклом: определяет очередного игрока, определяет окончание. Определяет победителя

Панель – содержит в себе модель и все элементы, влияющие на отображение игрового процесса. Отображает победившего игрока. Спрашивает у игрока размеры игрового поля при инициализации игры

Поле - квадратная область, состоящая из ячеек (пустых или содержащих буквы). Размеры поля задает игрок.

Ячейка - квадратная область поля. Одновременно может содержать в себе только одну букву.

Слово – последовательность букв, имеющая смысл и содержащаяся в словаре.

Словарь – сборник слов, которые можно составить из букв.

Обработчик счета – ведет подсчет очков, набираемых игроками за составление слов на игровом поле.

Словарь игроков – после окончания хода записывает в один из двух списков слов, составленное игроком слово

Визуальный алфавит – прямоугольная область, наполненная кнопками, с доступными буквами. Используется игроком для установки букв в ячейки.

3.4 Структура программы на уровне классов

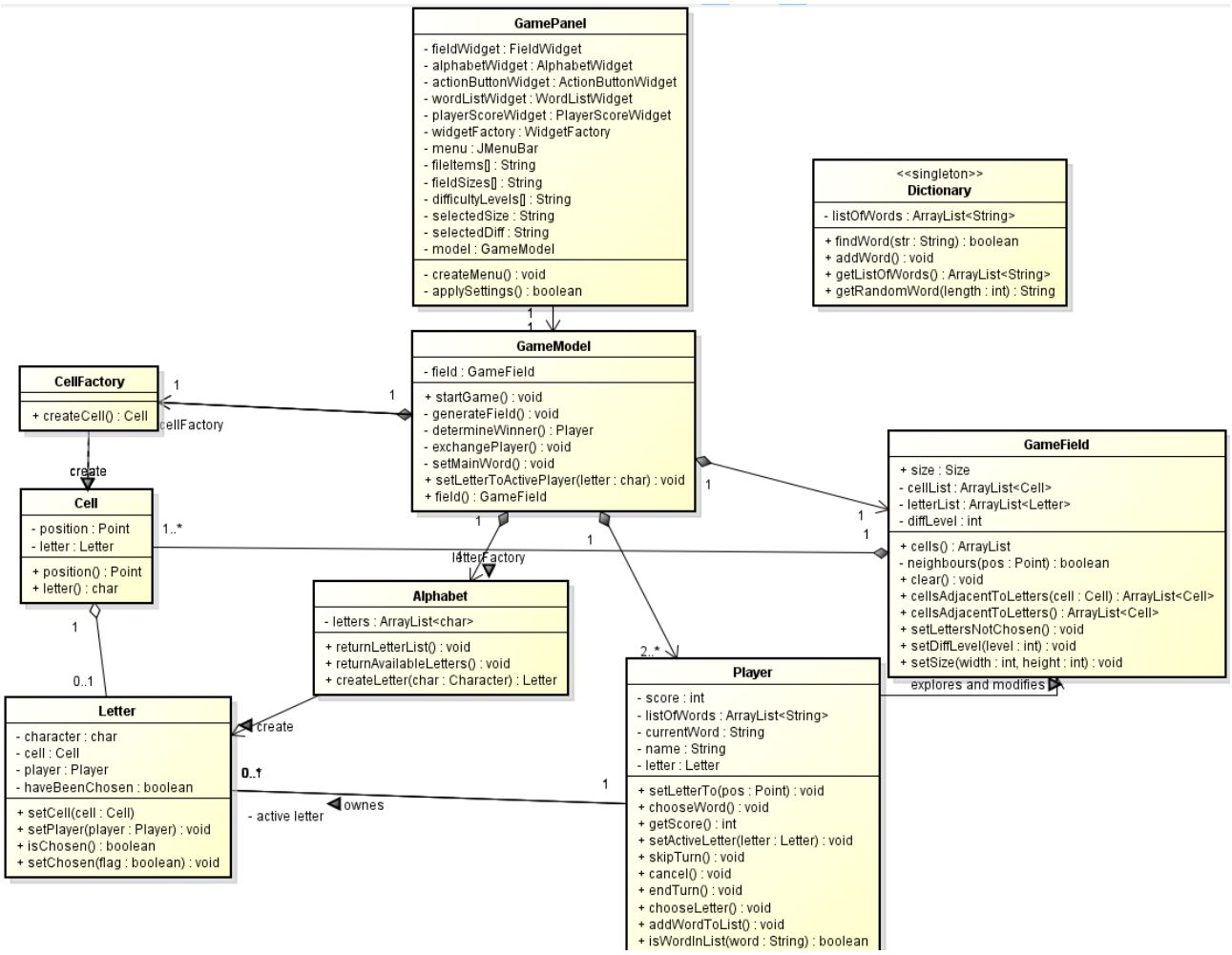


Диаграмма классов вычислительной модели

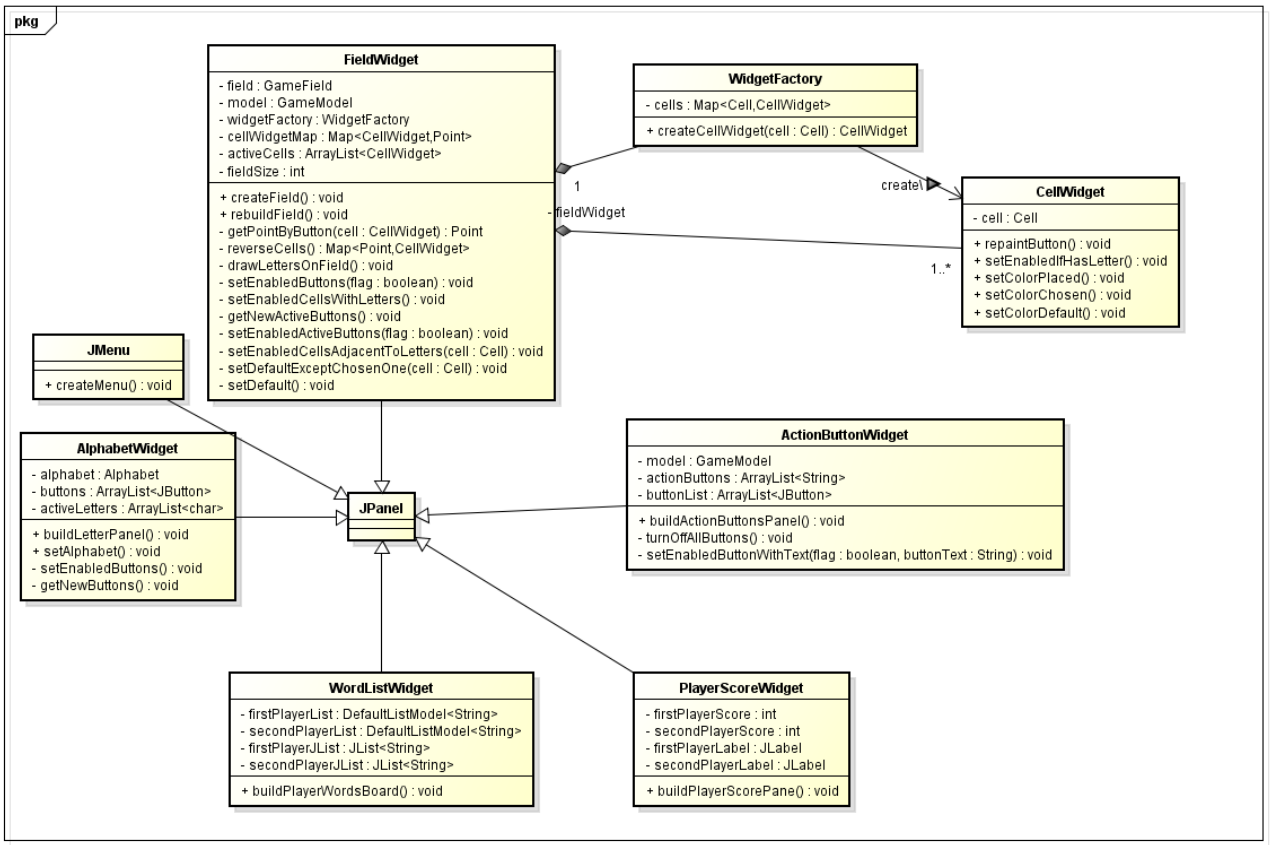
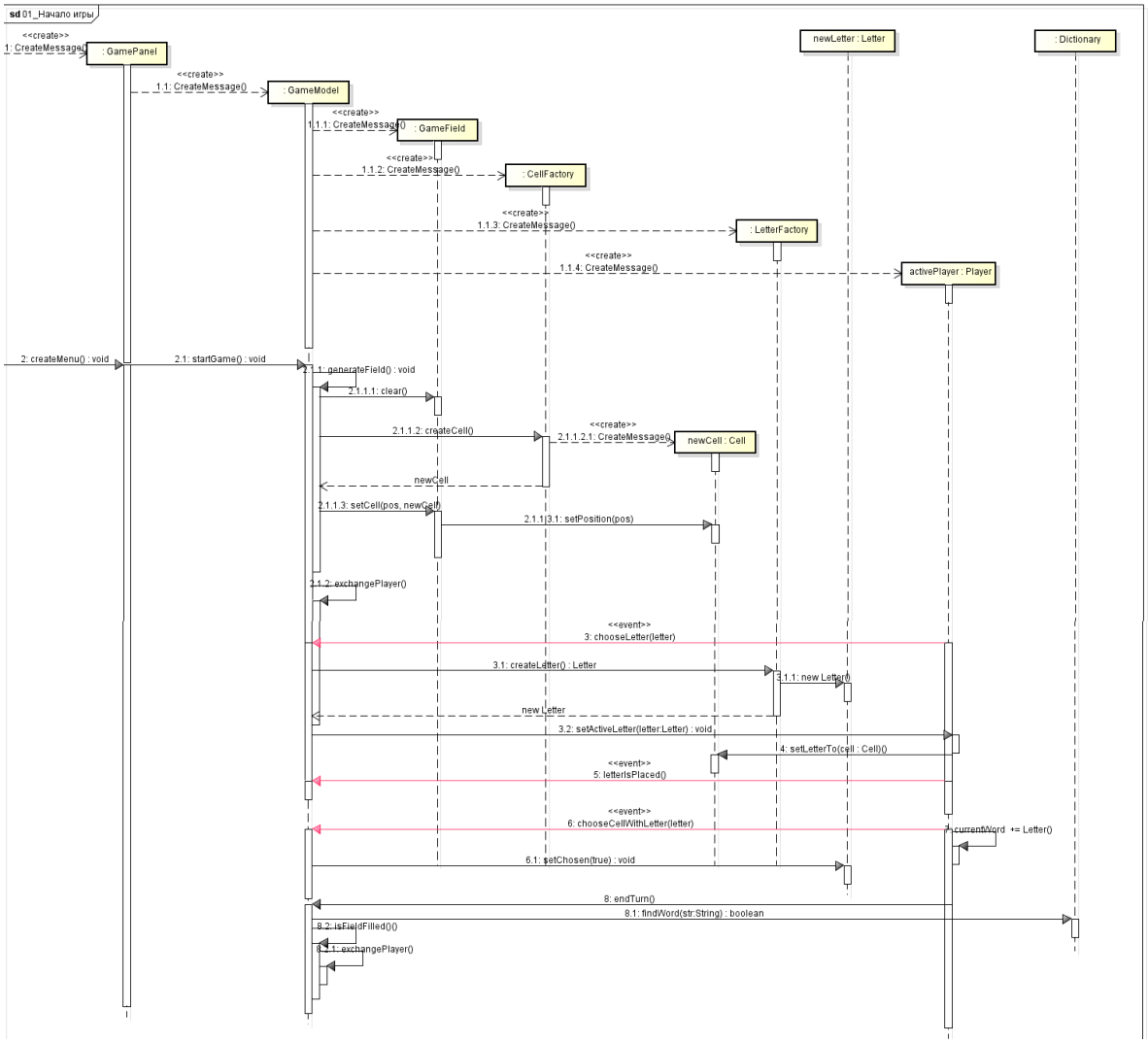
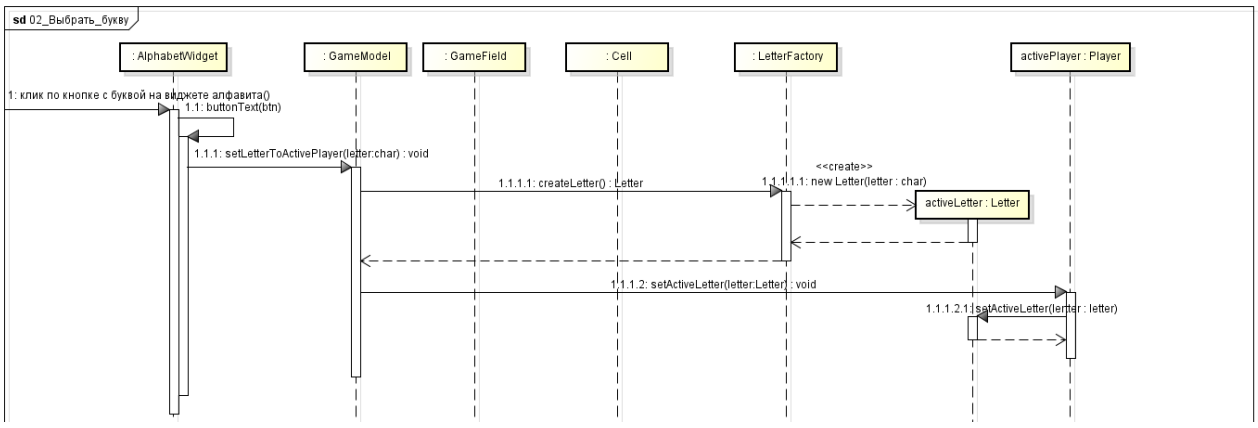


Диаграмма классов представления

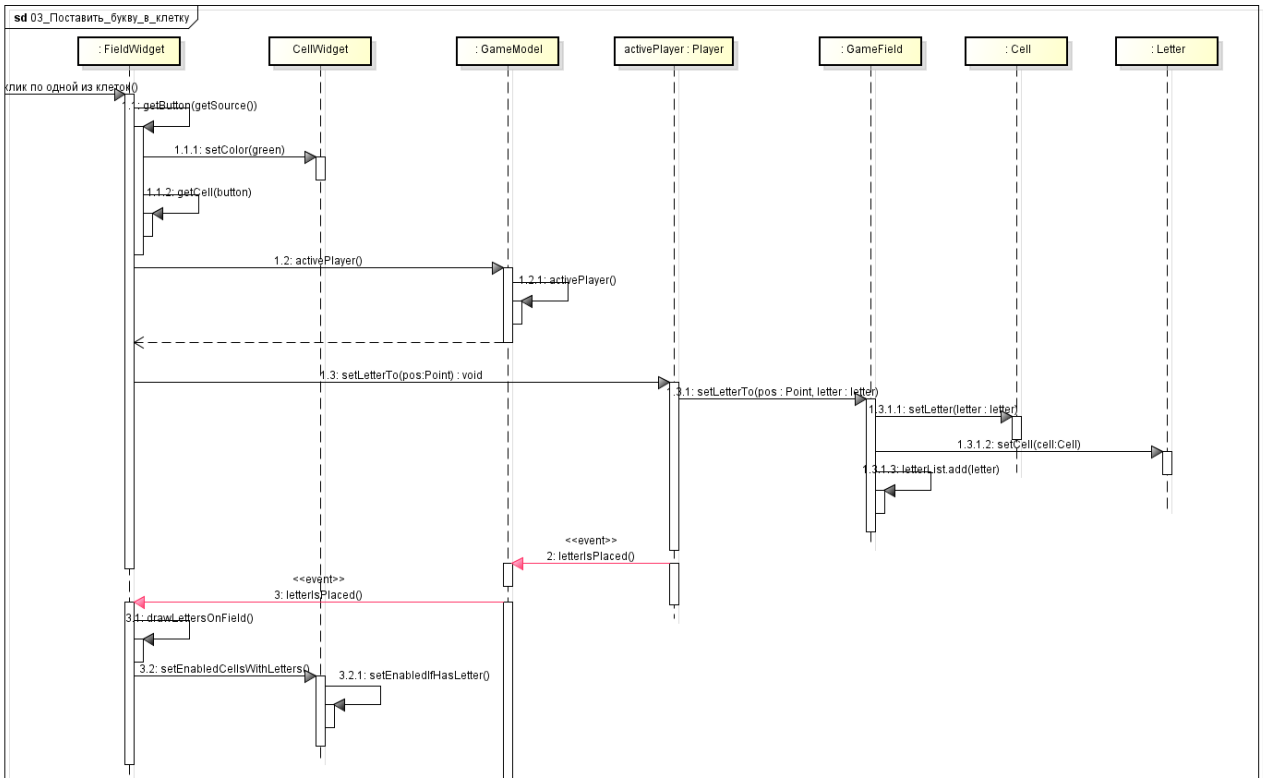
3.5 Типовые процессы в программе



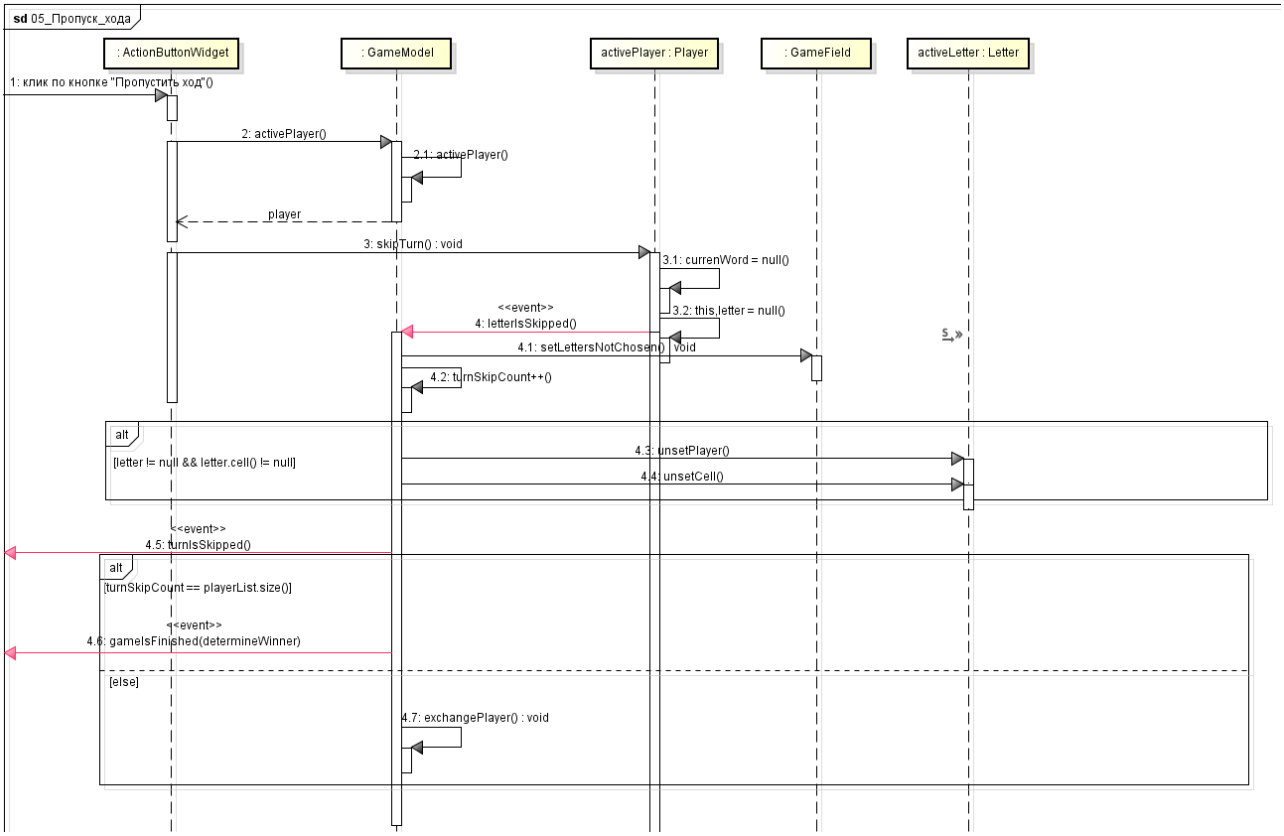
Общий игровой цикл



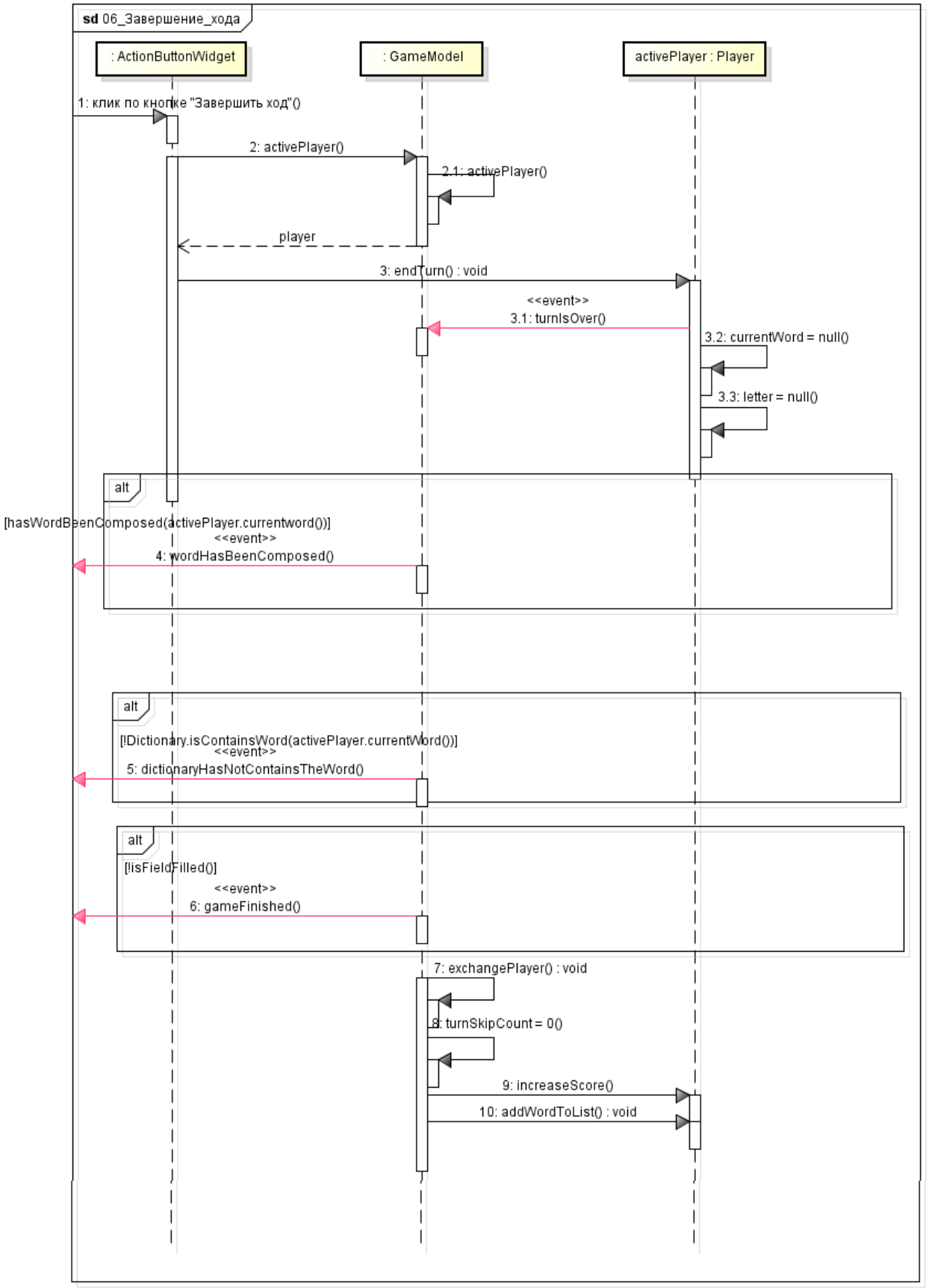
Выбор буквы



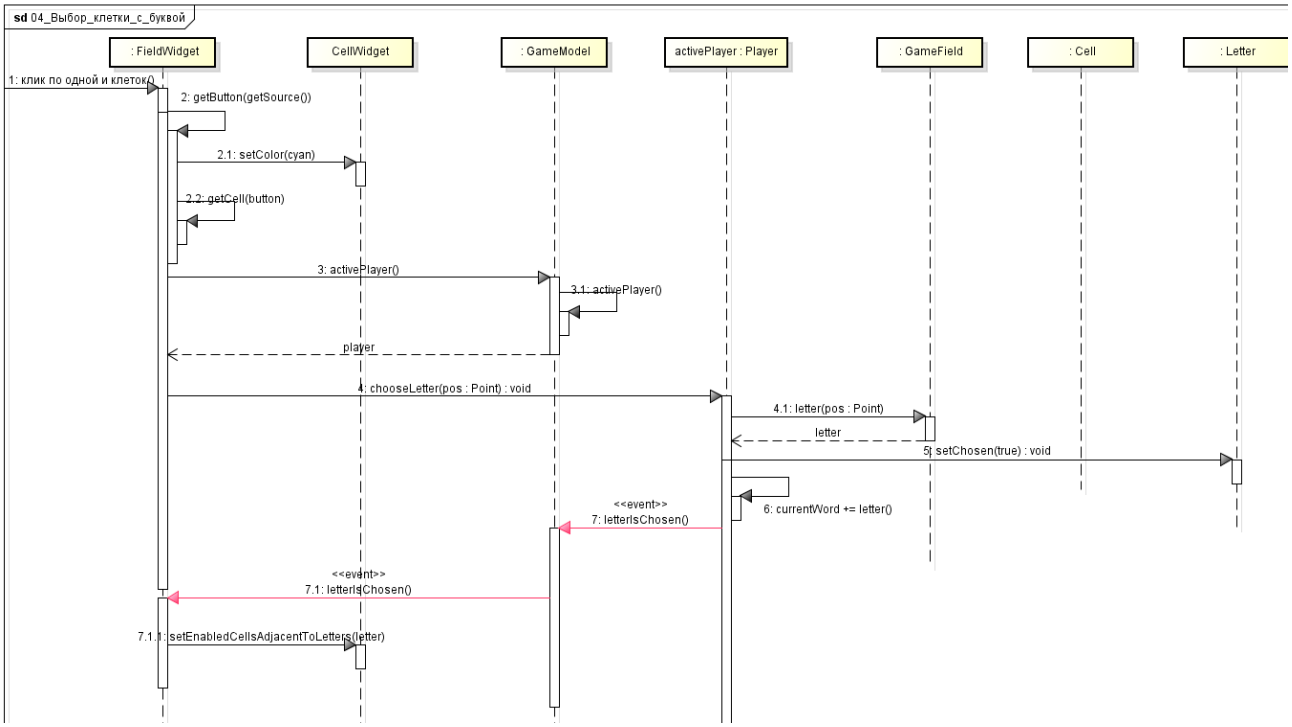
Установка буквы в клетку



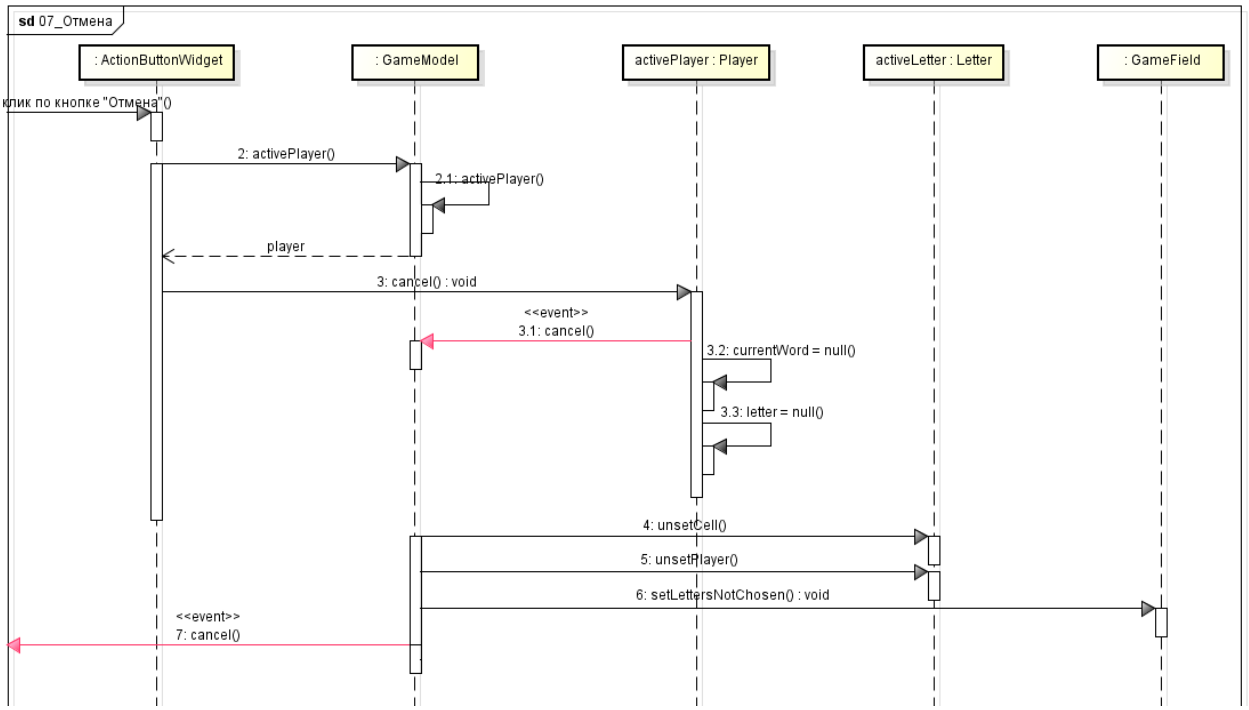
Пропуск хода



Завершение хода



Выбор клетки с буквой



Отмена

3.6 Человеко-машинное взаимодействие

Общий вид главного экрана программы представлен ниже. На нём располагается игровое поле, алфавит доступных букв, кнопки действий, счёт игроков, список слов, составленных игроками.

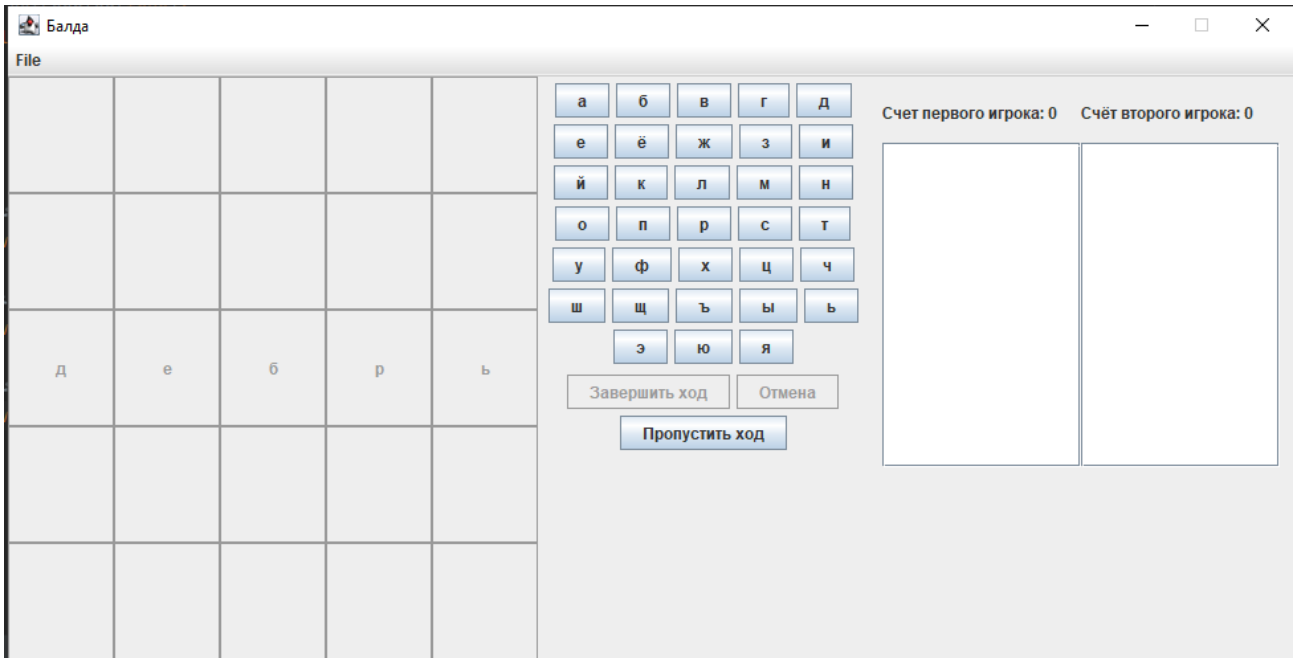


Рис. 1. Общий вид главного экрана программы

При инициализации новой игры появляется меню настроек игрового поля

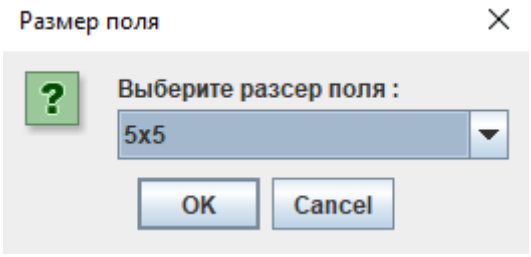


Рис. 2. Меню настроек

Всё управление игровым процессом происходит посредствам мыши.

3.7 Реализация ключевых классов

```
public class Alphabet{
    protected ArrayList<Character> letters = new
ArrayList<>(Arrays.asList(
            'а', 'б', 'в', 'г', 'д',
            'е', 'ё', 'ж', 'з', 'и',
            'й', 'к', 'л', 'м', 'н',
            'о', 'п', 'р', 'с', 'т',
            'у', 'ф', 'х', 'ц', 'ч',
            'ш', 'щ', 'ъ', 'ы', 'ь',
            'э', 'ю', 'я'));

    public List<Character> returnLetterList(){
        return Collections.unmodifiableList(letters);
    }

    public List<Character> returnAvailableLetters(){
        return Collections.unmodifiableList(letters);
    }
}


public class Cell {
    private final Point coordinates;

    public Cell(Point coordinates){
        this.coordinates = coordinates;
    }

    public Point position(){
        return (Point) coordinates.clone();
    }



    // --------- Клетка, принадлежит полю. Принадлежность задает
само поле ------
    private GameField field;

    void setField(GameField f){
        field = f;
    }

    // ---------------------- Буква, принадлежащая ячейке ----------
-------------

    private Letter letter;

    public void setLetter(Letter letter) {
        if(letter != null) {
            this.letter = letter;
            letter.setCell(this);
        }else
            this.letter = null;
```

```
    }

    public Letter letter(){
        return this.letter;
    }


    public boolean isAdjacentToLetterCell(){
        Point p = this.position();
        if (this.letter != null){
            return false;
        }
        if(field.cell(new Point(p.x + 1, p.y)) != null &&
field.cell(new Point(p.x + 1, p.y)).letter() != null){
            return true;
        } else if (field.cell(new Point(p.x - 1, p.y)) != null &&
field.cell(new Point(p.x - 1, p.y)).letter() != null) {
            return true;
        }else if (field.cell(new Point(p.x, p.y + 1)) != null &&
field.cell(new Point(p.x, p.y + 1)).letter() != null) {
            return true;
        }else if (field.cell(new Point(p.x, p.y - 1)) != null &&
field.cell(new Point(p.x, p.y - 1)).letter() != null) {
            return true;
        }
        return false;
    }
}


public class Dictionary {

    private static List<String> listOfWords = new ArrayList<>();

    private static Dictionary dictionary = new Dictionary();

    private Dictionary() {
        FileReader fr;
        try {
            fr = new
FileReader("\\Users\\Reversi\\java_projects\\buysell\\scrabble\\src
\\resources\\russian.txt");
            Scanner scanner = new Scanner(fr);
            while (scanner.hasNextLine()){
                listOfWords.add(scanner.nextLine());
            }
            fr.close();
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public static Dictionary getDictionary(){
```

```
        if(dictionary == null){
            dictionary = new Dictionary();
        }
        return dictionary;
    }

    public static List<String> getListOfWords(){
        return Collections.unmodifiableList(listOfWords);
    }

    public static void addWord(String word){
        FileWriter fw;
        try {
            fw = new
FileWriter("\\Users\\Reversi\\java_projects\\buysell\\scrabble\\src
\\resources\\russian.txt", true);
            fw.write("\n" + word);
            listOfWords.add(word);
            fw.close();
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public static boolean isContainsWord(String word){
        return listOfWords.contains(word);
    }

    public static String getRandomWord(int len) {
        Random random = new Random();
        String word =
listOfWords.get(random.nextInt(listOfWords.size()));
        while(word.length() != len){
            word =
listOfWords.get(random.nextInt(listOfWords.size()));
        }
        return word;
    }
}


package model.entity;

import java.awt.*;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class GameField {

    // ------------------------------ Клетки -----------------------
-----------------
    private ArrayList<Cell> cellList = new ArrayList<>();
```

```
public List<Cell> cells(){
    return Collections.unmodifiableList(cellList);
}

public Cell cell(Point pos){

    for(Cell obj : cellList)
    {
        if(obj.position().equals(pos))
        { return obj; }
    }

    return null;
}

void setCell(Cell cell){
    // Удаляем старую ячейку
    removeCell(cell.position());

    // Связываем ячейку с полем
    cell.setField(this);

    // Добавляем новую ячейку
    cellList.add(cell);
}

public void clear(){
    cellList.clear();
}

private void removeCell(Point pos){

    Cell obj = cell(pos);
    if(obj != null)     {
        cellList.remove(obj);
        obj.setField(null);
    }
}

//Возвращает все клетки, рядом с которыми есть клетка с буквой
public List<Cell> cellsAdjacentToLetters(){
    ArrayList<Cell> cells = new ArrayList<>();
    for(Cell c : cellList){
        if(c.isAdjacentToLetterCell())
            cells.add(c);
    }

    if (diffLevel == 2 && cells.size() > 2){
        int numberOfElementsToRemove = cells.size() / 3;

        Random random = new Random();

        for (int i = 0; i < numberOfElementsToRemove; i++){
            cells.remove(random.nextInt(cells.size()));
        }
```

```
        }
        return Collections.unmodifiableList(cells);
    }

    private List<Cell> neighbours(Cell cell){
        ArrayList<Cell> cells = new ArrayList<>();
        Point p = cell.position();
        if(cell(new Point(p.x + 1, p.y)) != null) {
                cells.add(cell(new Point(p.x + 1, p.y)));
        }if(cell(new Point(p.x - 1, p.y)) != null) {
                cells.add(cell(new Point(p.x - 1, p.y)));
        }if(cell(new Point(p.x, p.y + 1)) != null) {
                cells.add(cell(new Point(p.x, p.y + 1)));
        }if(cell(new Point(p.x, p.y - 1)) != null) {
                cells.add(cell(new Point(p.x, p.y - 1)));
        }
        return cells;
    }

    //Возвращает клетки-соседи с переданной, в которых есть буква,
и которые еще не были выбраны
    public List<Cell> cellsAdjacentToLetters(Cell cell){
        ArrayList<Cell> cells = new ArrayList<>();
        for(Cell c : neighbours(cell)){
            if(c.letter() != null && !c.letter().isChosen()){
                cells.add(c);
            }
        }

        return Collections.unmodifiableList(cells);
    }

    public void setLettersNotChosen(){
        for (Letter l : letterList){
            l.setChosen(false);
        }
    }

    // ------------------------------ Буквы ------------------------
----------------

    private ArrayList<Letter> letterList = new ArrayList<>();

    public List<Letter> letters() {
        letterList.clear();
        for (Cell c : cellList){
            Letter l = c.letter();
            if (l != null)
            letterList.add(c.letter());
        }
        return Collections.unmodifiableList(letterList);
    }

    public Letter letter(Point pos){
        Cell obj = cell(pos);
        if(obj != null) return obj.letter();
```

```
        return null;
    }

    public void setLetter(Point pos, Letter letter){
        Cell obj = cell(pos);
        if(obj != null){
            obj.setLetter(letter);
            letter.setCell(obj);
            letterList.add(letter);
        }
    }

    private int diffLevel = 2;

    public void setDiffLevel(int level) {
        if(level != 1 && level != 2) throw new RuntimeException("В
поле передан неправильный уровень сложности!");
        diffLevel = level;
    }

    // ----------------------- Ширина и высота поля ---------------
---------------

    public GameField(){
        setSize(5, 5);
    }
    private int width;
    private int height;


    public void setSize(int width, int height) {

        this.width = width;
        this.height = height;

        // Удаляем все ячейки находящиеся вне поля
        ArrayList<Cell> removableList = new ArrayList<>();
        for (Cell obj : cellList) {
            if(!containsRange(obj.position())) {
                removableList.add(obj);
            }
        }
        for (Cell c : removableList) {
            cellList.remove(c);
        }
    }

    public int width() {
        return width;
    }

    public int height() {
        return height;
    }

    public boolean containsRange(Point p){
```

```
        return p.getX() >= 0 && p.getX() < width &&
                p.getY() >= 0 && p.getY() < height;
    }
}
```

```
package model.entity;

import model.events.GameEvent;
import model.events.GameListener;
import model.events.PlayerActionEvent;
import model.events.PlayerActionListener;
import model.factory.CellFactory;
import model.factory.LetterFactory;

import java.awt.*;
import java.util.ArrayList;

public class GameModel {

    // ---------------------------------- Поле ----------------------
---------------
    private GameField field = new GameField();

    public GameField field() {
        return this.field;
    }

    // ---------------------------------- Игроки --------------------
---------------
    private ArrayList<Player> playerList = new ArrayList<>();
    private int activePlayer;

    private int turnSkipCount = 0;
    public Player activePlayer(){
        return playerList.get(activePlayer);
    }

    public GameModel(){
        //Размеры поля по умолчанию
        field.setSize(8, 8);

        //Создание игроков
        Player p;
        PlayerObserver observer = new PlayerObserver();

        p = new Player(field(), "1", this);
        p.addPlayerActionListener(observer);
        playerList.add(p);
        activePlayer = 0;

        p = new Player(field(), "2", this);
        p.addPlayerActionListener(observer);
        playerList.add(p);
    }
```

```
    // ---------------------- Порождение обстановки на поле -------
--------------

    private CellFactory cellFactory = new CellFactory();

    private void generateField(){

        field().clear();
        for(int row = 0; row < this.field().height(); row++) {
            for(int col = 0; col < this.field().width(); col++) {
                field().setCell(cellFactory.createCell(new
Point(col, row)));
            }
        }
    }

    // ----------------------------- Игровой процесс --------------
--------------

    public void startGame(){
        //Генерируем поле
        generateField();
        setMainWord();

        //Передаем ход первому игроку
        activePlayer = playerList.size()-1;
        exchangePlayer();
    }

    private LetterFactory letterFactory = new LetterFactory();
    public void exchangePlayer(){
        activePlayer++;
        if(activePlayer >= playerList.size()) activePlayer = 0;

        field.setLettersNotChosen();

        if(isFieldFilled()){
            fireGameFinished(determineWinner());
        }
    }

    private void setMainWord() {
        String word = Dictionary.getRandomWord(field().width());
        for (int i = 0; i < field().width(); i++){
            field().setLetter(new Point(i, field().height()/2),
letterFactory.createLetter(word.charAt(i)));
        }
    }
    public void setLetterToActivePlayer(char ch){

activePlayer().setActiveLetter(letterFactory.createLetter(ch));
    }

    private boolean isFieldFilled(){
        for (Cell cell : field.cells()){
            if(cell.letter() == null){
```

```
                return false;
            }
        }
        return true;
    }

    private boolean hasWordBeenComposed(String word){
        for (Player p : playerList){
            boolean flag = p.isWordInList(word);
            if(flag == true)
                return true;
        }
        return false;
    }

    public Player determineWinner(){
        turnSkipCount = 0;
        if (playerList.get(0).score() > playerList.get(1).score()){
            return playerList.get(0);
        }
        else if (playerList.get(0).score() <
playerList.get(1).score()) {
            return playerList.get(1);
        }
        else return null;
    }

    // ------------------------- Реагируем на действия игрока -----
-------------

    private class PlayerObserver implements PlayerActionListener {

        @Override
        public void letterIsPlaced(PlayerActionEvent e) {
            if (e.player() == activePlayer()) {
                fireLetterIsPlaced(e);
            }
        }

        @Override
        public void letterIsReceived(PlayerActionEvent e) {
            if (e.player() == activePlayer()) {
                fireLetterIsReceived(e);
            }
        }

        @Override
        public void turnIsSkipped(PlayerActionEvent e) {
            if (e.letter() != null) {
                if (e.letter().cell() != null) {
                    Letter l = e.letter();
                    l.unsetCell();
                    l.unsetPlayer();
                }
            }
            if (e.player() == activePlayer()) {
```

```
                fireTurnIsSkipped(e);
            }

            turnSkipCount++;
            field.setLettersNotChosen();

            if (turnSkipCount == playerList.size()) {
                fireGameFinished(determineWinner());
            } else {
                exchangePlayer();
            }
        }

        @Override
        public void letterOnFieldIsChosen(PlayerActionEvent e) {
            if (e.player() == activePlayer()) {
                fireLetterOnFieldIsChosen(e);
            }

            if (e.letter() == e.player().activeLetter()) {
                fireCurrentLetterIsChosen(e.player());
            }
        }

        @Override
        public void turnIsOver(PlayerActionEvent e) {
            if (hasWordBeenComposed(e.player().currentWord())) {
                fireWordHasBeenComposed(activePlayer());
                return;
            }
            if
(!Dictionary.isContainsWord(e.player().currentWord())) {
                fireDictionaryHasNotContainsWord(e.player());
                return;
            }

            if (e.player() == activePlayer()) {
                fireTurnIsOver(e);
            }

            if (!isFieldFilled()) {
                exchangePlayer();
                turnSkipCount = 0;
                e.player().increaseScore(e.currentWord().length());
                e.player().addWordToList();
                System.out.println("Игрок: " + e.player().name() +
" добавил слово " + e.player().currentWord() + " Счёт игрока:" +
e.player().score());

            } else {
                fireGameFinished(determineWinner());
            }
        }

        @Override
        public void cancel(PlayerActionEvent e) {
```

```
            Letter l = e.letter();
            l.unsetCell();
            l.unsetPlayer();
            fireCancel(e);
            field.setLettersNotChosen();
        }
    }

        // ------------------------ Порождает события игры --------
--------------------

        private ArrayList<GameListener> gameListenerList = new
ArrayList<>();

        public void addGameListener(GameListener l) {
            gameListenerList.add(l);
        }

        public void removeGameListener(GameListener l) {
            gameListenerList.remove(l);
        }

        protected void fireGameFinished(Player winner) {
            GameEvent event = new GameEvent(this);
            event.setPlayer(winner);
            for (GameListener listener : gameListenerList) {
                listener.gameFinished(event);
            }
        }

        protected void fireCurrentLetterIsChosen(Player p) {
            GameEvent event = new GameEvent(this);
            event.setPlayer(p);
            for (GameListener listener : gameListenerList) {
                listener.currentLetterIsChosen(event);
            }
        }

    protected void fireDictionaryHasNotContainsWord(Player p) {
        GameEvent event = new GameEvent(this);
        event.setPlayer(p);
        for (GameListener listener : gameListenerList) {
            listener.dictionaryHasNotContainsWord(event);
        }
    }

    protected void fireWordHasBeenComposed(Player p) {
        GameEvent event = new GameEvent(this);
        event.setPlayer(p);
        for (GameListener listener : gameListenerList) {
            listener.wordHasBeenComposed(event);
        }
    }

        // --------------------- Порождает события, связанные с
игроками -------------
```

```
    private ArrayList<PlayerActionListener>
playerActionListenerList = new ArrayList<>();

    public void addPlayerActionListener(PlayerActionListener l) {
        playerActionListenerList.add(l);
    }

    public void removePlayerActionListener(PlayerActionListener l)
{
        playerActionListenerList.remove(l);
    }

    protected void fireLetterIsPlaced(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.letterIsPlaced(e);
        }
    }

    protected void fireLetterIsReceived(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.letterIsReceived(e);
        }
    }

    protected void fireTurnIsSkipped(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.turnIsSkipped(e);
        }
    }

    protected void fireLetterOnFieldIsChosen(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.letterOnFieldIsChosen(e);
        }
    }

    protected void fireTurnIsOver(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.turnIsOver(e);
        }
    }

    protected void fireCancel(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.cancel(e);
        }
    }
}
```

```java
public class Letter {

    // --------------------- Буква -----------------------
    private final char character;

    public Letter(char ch){
        this.character = ch;
    }

    public char character(){
        return this.character;
    }

    // --------------------- Клетка -----------------------
    private Cell cell = null;

    public Cell cell(){ return this.cell; }

    public void setCell(Cell cell) {
        if(this.cell == null){
            this.cell = cell;
            cell.setLetter(this);
        }
    }

    public void unsetCell(){
        this.cell.setLetter(null);
        this.cell = null;
    }

    //Была ли буква выбрана
    private boolean haveBeenChosen = false;

    public boolean isChosen() {
        return haveBeenChosen;
    }

    public void setChosen(boolean haveChosen) {
        this.haveBeenChosen = haveChosen;
    }

    // Игрок, которому прнадлежит буква. Буква может быть
нейтральной (не принадлежать никому) -

    private Player player = null;

    public Player player(){ return this.player; }

    public void setPlayer(Player p){ this.player = p; }

    public void unsetPlayer(){
        this.player = null;
    }
}
```

```
public class Player {
    // ------------------------------- Поля связанные с игроком -
------------------------------
    private int playerScore = 0;

    private String name;

    public String name(){
        return this.name;
    }

    private String currentWord = "";

    public String currentWord(){
        return currentWord;
    }


    // ----------------------- Устанавливаем связь с полем --------
---------------
    GameField field;

    public Player (GameField field, String name, GameModel model) {
        model.addGameListener(new GameObserver());
        this.field = field;
        this.name = name;
    }

    // ---------------------- Счёт игрока -----------------------

    public int score(){
        return playerScore;
    }

    public void increaseScore(int points){
        playerScore += points;
    }

    // ---------------------- Слова, составленные игроком ---------
--------------
    private ArrayList<String> words = new ArrayList<>();

    public boolean isWordInList(String word){
        return words.contains(word);
    }

    public void addWordToList(){
        if(currentWord.length() > 0)
            words.add(currentWord);
    }


    // ---------------------- Буква, которую нужно установить -----
------------------
```

```
    private Letter letter;

    public void setActiveLetter(Letter l) {
        this.letter = l;
        l.setPlayer(this);

        // Генерируем событие
        fireLetterIsReceived(this.letter);
    }

    public Letter activeLetter() {
        return this.letter;
    }

    public void setLetterTo(Point pos){

        if (this.letter == null) throw new
IllegalArgumentException("Буква не была задана!");

        this.field.setLetter(pos, this.letter);

        // Генерируем событие
        if(this.field.letter(pos) == this.letter) {
            fireLetterIsPlaced(this.letter);
            //this.letter = null;
        }
    }

    // ---------------------- Процесс игры -----------------------

    public void chooseLetter(Point pos){
        Letter chosenLetter = field.letter(pos);
        if(chosenLetter == null) throw new FindException("В
выбранной клетке нет буквы!");

        chosenLetter.setChosen(true);
        currentWord += String.valueOf(chosenLetter.character());

        fireLetterOnFieldIsChosen(chosenLetter);
    }

    public void skipTurn(){
        fireTurnIsSkipped(this.letter);
        currentWord = "";
        this.letter = null;
    }

    public void endTurn(){
        fireTurnIsOver(this.letter, this.currentWord);
        currentWord = "";
        this.letter = null;
    }

    public void cancel(){
        fireCancel(this.letter, this.currentWord);
        currentWord = "";
```

```
        this.letter = null;
    }

    // ---------------------- Порождает события -------------------
----------

   private ArrayList<PlayerActionListener> listeners = new
ArrayList<>();

    // Присоединяет слушателя
    public void addPlayerActionListener(PlayerActionListener l) {
        this.listeners.add(l);
    }

    // Отсоединяет слушателя
    public void removePlayerActionListener(PlayerActionListener l)
{
        this.listeners.remove(l);
    }

    // Оповещает слушателей о событии
    protected void fireLetterIsPlaced(Letter l) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        for(PlayerActionListener o : listeners){
            o.letterIsPlaced(event);
        }
    }

    protected void fireLetterIsReceived(Letter l) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        for(PlayerActionListener o : listeners){
            o.letterIsReceived(event);
        }
    }

    protected void fireTurnIsSkipped(Letter l) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        for (PlayerActionListener o : listeners){
            o.turnIsSkipped(event);
        }
    }

    protected void fireLetterOnFieldIsChosen(Letter l) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        for (PlayerActionListener o : listeners){
            o.letterOnFieldIsChosen(event);
        }
    }
```

```
    protected void fireTurnIsOver(Letter l, String word) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        event.setCurrentWord(word);
        for (PlayerActionListener o : listeners){
            o.turnIsOver(event);
        }
    }

    protected void fireCancel(Letter l, String word) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        event.setCurrentWord(word);
        for (PlayerActionListener o : listeners){
            o.cancel(event);
        }
    }

    private class GameObserver implements GameListener{

        @Override
        public void gameFinished(GameEvent e) {
            playerScore = 0;
        }
        @Override
        public void currentLetterIsChosen(GameEvent e) {}
        @Override
        public void dictionaryHasNotContainsWord(GameEvent e) {}
        @Override
        public void wordHasBeenComposed(GameEvent e) {}
    }
}


public class GameEvent extends EventObject {

    Player player;

    public void setPlayer(Player player) {
        this.player = player;
    }

    public Player player() {
        return player;
    }

    public GameEvent(Object source){
        super(source);
    }
}


public interface GameListener extends EventListener {
```

```
    public void gameFinished(GameEvent e);

    public void currentLetterIsChosen(GameEvent e);

    public void dictionaryHasNotContainsWord(GameEvent e);

    public void wordHasBeenComposed(GameEvent e);
}


public interface MenuListener extends EventListener {
    public void newGameStarted();
}



public class PlayerActionEvent extends EventObject {
    // ------------------------------- Игрок --------------------
-----------------
    Player player;

    public void setPlayer(Player player) {
        this.player = player;
    }

    public Player player() {
        return player;
    }

    // ------------------------------- Активная буква ------------
-------------------------
    Letter letter;

    public void setLetter(Letter letter) {
        this.letter = letter;
    }

    public Letter letter() {
        return letter;
    }

    // ------------------------------- Составленное слово --------
-----------------------------
    String currentWord;

    public void setCurrentWord(String currentWord) {
        this.currentWord = currentWord;
    }

    public String currentWord() {
        return currentWord;
    }

    public PlayerActionEvent(Object source) {
        super(source);
```

```
    }
}


public interface PlayerActionListener extends EventListener {
    void letterIsPlaced(PlayerActionEvent e);

    void letterIsReceived(PlayerActionEvent e);

    void turnIsSkipped(PlayerActionEvent e);

    void letterOnFieldIsChosen(PlayerActionEvent e);

    void turnIsOver(PlayerActionEvent e);

    void cancel(PlayerActionEvent e);
}


public class CellFactory {
    public Cell createCell(Point point){
        return new Cell(point);
    }
}


public class LetterFactory {
    public Letter createLetter(char ch){
        return new Letter(ch);
    }
}


public class ActionButtonWidget extends JPanel {
    private GameModel model;

    private List<String> actionButtons = Arrays.asList("Завершить
ход", "Отмена", "Пропустить ход");

    private List<JButton> buttonList = new ArrayList<>();

    public ActionButtonWidget(GameModel model, GamePanel panel){
        model.addPlayerActionListener(new PlayerObserver());
        model.addGameListener(new GameObserver());
        panel.addMenuListener(new MenuObserver());
        this.model = model;
    }

    public void buildActionButtonsPanel(){
        setLayout(new FlowLayout());
        Dimension dimension = new Dimension(250, 40);

        setPreferredSize(dimension);

        JButton btn;
```

```
        for (String str : actionButtons){
            btn = new JButton(str);
            add(btn);
            btn.addActionListener(new ActionButtonClickListener());
            buttonList.add(btn);
            btn.setEnabled(false);
        }
    }

    private void turnOffAllButtons(){
        for (JButton button : buttonList){
            button.setEnabled(false);
        }
    }

    private void setEnabledButtonWithText(boolean flag, String
buttonText){
        for (JButton button : buttonList){
            if(button.getText().equals(buttonText)){
                button.setEnabled(flag);
            }
        }
    }

    private class ActionButtonClickListener implements
ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            JButton btn = (JButton) e.getSource();
            String btnText = btn.getText();
            System.out.println(btnText);
            if (btnText.equals("Пропустить ход")) {
                model.activePlayer().skipTurn();
            } else if (btnText.equals("Завершить ход")) {
                model.activePlayer().endTurn();
            } else if (btnText.equals("Отмена")) {
                model.activePlayer().cancel();
            }
        }
    }

    private class PlayerObserver implements PlayerActionListener{

        @Override
        public void letterIsPlaced(PlayerActionEvent e) {
            setEnabledButtonWithText(true, "Отмена");
        }

        @Override
        public void letterIsReceived(PlayerActionEvent e) {

        }

        @Override
        public void turnIsSkipped(PlayerActionEvent e) {
            setEnabledButtonWithText(false, "Завершить ход");
```

```
                setEnabledButtonWithText(false, "Отмена");
            }

            @Override
            public void letterOnFieldIsChosen(PlayerActionEvent e) {

            }

            @Override
            public void turnIsOver(PlayerActionEvent e) {
                setEnabledButtonWithText(false, "Завершить ход");
                setEnabledButtonWithText(false, "Отмена");
            }

            @Override
            public void cancel(PlayerActionEvent e) {
                setEnabledButtonWithText(false, "Завершить ход");
                setEnabledButtonWithText(false, "Отмена");
            }
        }

        private class GameObserver implements GameListener{

            @Override
            public void gameFinished(GameEvent e) {
                turnOffAllButtons();
            }

            @Override
            public void currentLetterIsChosen(GameEvent e) {
                setEnabledButtonWithText(true, "Завершить ход");
            }

            @Override
            public void dictionaryHasNotContainsWord(GameEvent e) {

            }

            @Override
            public void wordHasBeenComposed(GameEvent e) {

            }
        }

        private class MenuObserver implements MenuListener {
            @Override
            public void newGameStarted() {
                turnOffAllButtons();
                setEnabledButtonWithText(true, "Пропустить ход");
            }
        }
    }


    public class AlphabetWidget extends JPanel {
        private GameModel model;
```

```
    private ComplicatedAlphabet alphabet;

    private ArrayList<JButton> buttonList = new ArrayList<>();

    private List<Character> activeLetters;

    public AlphabetWidget(ComplicatedAlphabet alphabet, GameModel
model, GamePanel panel){
        model.addGameListener(new GameObserver());
        model.addPlayerActionListener(new PlayerObserver());
        panel.addMenuListener(new MenuObserver());
        this.model = model;
        this.alphabet = alphabet;
    }

    public void buildLetterPanel(){
        List<Character> letters;
        letters = alphabet.returnLetterList();

        setLayout(new FlowLayout());
        Dimension dimension = new Dimension(250, 220);
        setPreferredSize(dimension);
        JButton btn;
        for (Character letter : letters){
            btn = new JButton(Character.toString(letter));
            add(btn);
            btn.addActionListener(new LetterClickListener());
            buttonList.add(btn);
            btn.setEnabled(false);
        }
        activeLetters = alphabet.returnAvailableLetters();
    }

    public void setAlphabet(ComplicatedAlphabet alphabet){
        this.alphabet = alphabet;
    }

    private void setEnabledButtons(boolean flag){
        for (Character ch : activeLetters){
            for (JButton button : buttonList){
                if(button.getText().equals(ch.toString()))
                    button.setEnabled(flag);
            }
        }
    }

    private void setEnabledAllButtons(boolean flag){
        for (JButton button : buttonList){
            button.setEnabled(flag);
        }
    }

    private void getNewButtons(){
        activeLetters = alphabet.returnAvailableLetters();
    }
```

```
    private class LetterClickListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            JButton btn = (JButton) e.getSource();
            String btnText = btn.getText();
            System.out.println(btnText);
            model.setLetterToActivePlayer(btnText.charAt(0));
        }
    }

    private class PlayerObserver implements PlayerActionListener{
        @Override
        public void letterIsPlaced(PlayerActionEvent e) {
            setEnabledAllButtons(false);
        }
        @Override
        public void letterIsReceived(PlayerActionEvent e) {}

        @Override
        public void turnIsSkipped(PlayerActionEvent e) {
            getNewButtons();
            setEnabledButtons(true);
        }
        @Override
        public void letterOnFieldIsChosen(PlayerActionEvent e) {}

        @Override
        public void turnIsOver(PlayerActionEvent e) {
            getNewButtons();
            setEnabledButtons(true);
        }

        @Override
        public void cancel(PlayerActionEvent e) {
            setEnabledButtons(true);
        }
    }

    private class GameObserver implements GameListener{
        @Override
        public void gameFinished(GameEvent e) {
            setEnabledAllButtons(false);
        }
        @Override
        public void currentLetterIsChosen(GameEvent e) {}
        @Override
        public void dictionaryHasNotContainsWord(GameEvent e) {}
        @Override
        public void wordHasBeenComposed(GameEvent e) {}
    }

    private class MenuObserver implements MenuListener{
        @Override
        public void newGameStarted() {
            setEnabledAllButtons(false);
```

```
                getNewButtons();
                setEnabledButtons(true);
            }
        }
    }


public class CellWidget extends JButton {

    private Cell cell;


    public CellWidget(Cell cell){
        if(cell == null) throw new RuntimeException("В CellWidget
передан null");
        else{
            this.cell = cell;
            if(cell.letter() != null)
                setText(String.valueOf(cell.letter().character()));
            //setBackground(Color.WHITE);

            setFocusable(false);
            setVisible(true);
        }
    }

    public void repaintButton(){
        if(cell.letter() != null){
            setText(String.valueOf(cell.letter().character()));
        } else {
            setText(null);
        }
        revalidate();
        repaint();
    }

    public void setEnabledIfHasLetter(){
        if (cell.letter() != null){
            setEnabled(true);
        } else {
            //setColorDefault();
            setEnabled(false);
        }
    }

    public void setColorPlaced(){
        setBackground(Color.GREEN);
    }

    public void setColorChosen(){
        setBackground(Color.CYAN);
    }

    public void setColorDefault(){
        setBackground(null);
```

```
    }
}


public class FieldWidget extends JPanel {
    private GameModel model;

    private GameField field;

    private WidgetFactory factory = new WidgetFactory();

    private final Map<CellWidget, Point> cellWidgetMap = new
HashMap<>();

    private ArrayList<CellWidget> activeCells = new ArrayList<>();

    private final int FIELD_SIZE = 400;

    public FieldWidget(GameField field, GameModel model, GamePanel
panel) {
        model.addPlayerActionListener(new PlayerObserver());
        model.addGameListener(new GameObserver());
        panel.addMenuListener(new MenuObserver());
        this.model = model;
        this.field = field;
    }

    public void createField() {
        setLayout(new GridLayout(field.height(), field.width()));

        Dimension dimension = new Dimension(FIELD_SIZE,
FIELD_SIZE);

        setPreferredSize(dimension);
        setMinimumSize(dimension);
        setMaximumSize(dimension);

        setVisible(true);
    }

    public void rebuildField() {
        removeAll();
        cellWidgetMap.clear();

        setLayout(new GridLayout(field.height(), field.height()));

        for (int row = 0; row < field.height(); row++) {
            for (int col = 0; col < field.width(); col++) {
                Point p = new Point(col, row);
                CellWidget button =
factory.createCellWidget(field.cell(p));
                button.addActionListener(new FieldClickListener());
                add(button);
                cellWidgetMap.put(button, p);
            }
        }
```

```
        validate();
    }

    private Point getPointByButton(CellWidget cellWidget){
        return (Point) cellWidgetMap.get(cellWidget).clone();
    }

    private Map<Point, CellWidget> reverseCells() {
        HashMap<Point, CellWidget> reverseMap = new HashMap<>();
        for (Map.Entry<CellWidget, Point> entry :
cellWidgetMap.entrySet()) {
            reverseMap.put(entry.getValue(), entry.getKey());
        }
        return reverseMap;
    }

    private void drawLettersOnField() {
        for (Map.Entry<CellWidget, Point> entry :
cellWidgetMap.entrySet()) {
            entry.getKey().repaintButton();
        }
    }

    private void setEnabledButtons(boolean flag) {
        for (CellWidget cell : cellWidgetMap.keySet()) {
            cell.setEnabled(flag);
        }
    }

    private void setEnabledCellsWithLetters() {
        for (Map.Entry<CellWidget, Point> entry :
cellWidgetMap.entrySet()) {
            entry.getKey().setEnabledIfHasLetter();
        }
    }

    private void getNewActiveButtons() {
        setEnabledButtons(false);
        activeCells.clear();
        for (Cell c : field.cellsAdjacentToLetters()) {
            CellWidget widget = reverseCells().get(c.position());
            activeCells.add(widget);
        }
    }

    private void setEnabledActiveButtons(boolean flag){
        for (CellWidget w : activeCells){
            w.setEnabled(flag);
        }
    }

    private void setEnabledCellsAdjacentToLetters(Cell cell) {
        Map<Point, CellWidget> map = reverseCells();
        for (Cell c : field.cellsAdjacentToLetters(cell)) {
            map.get(c.position()).setEnabled(true);
        }
```

```
    }

    private void setDefaultExceptChosenOne(Cell cell){
        for (Map.Entry<CellWidget, Point> entry :
cellWidgetMap.entrySet()) {
            if (!entry.getValue().equals(cell.position())) {
                entry.getKey().setColorDefault();
            }
            else entry.getKey().setColorPlaced();
        }
    }

    private void setDefault(){
        for (Map.Entry<CellWidget, Point> entry :
cellWidgetMap.entrySet()) {
            entry.getKey().setColorDefault();
        }
    }

    private class FieldClickListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {

            CellWidget button = (CellWidget) e.getSource();
            button.setEnabled(false);

            // Ставим на поле букву текущего игрока
            Point p = getPointByButton(button);
            for (Cell c : field.cells()) {
                if (c.position().equals(p)) {
                    if (c.letter() == null) {
                        button.setColorPlaced();
                        model.activePlayer().setLetterTo(p);
                    } else {
                        button.setColorChosen();
                        model.activePlayer().chooseLetter(p);
                    }
                }
            }
        }
    }

    private class PlayerObserver implements PlayerActionListener{

        @Override
        public void letterIsPlaced(PlayerActionEvent e) {
            drawLettersOnField();
            setEnabledCellsWithLetters();
            setDefaultExceptChosenOne(e.letter().cell());
        }

        @Override
        public void letterIsReceived(PlayerActionEvent e) {
            setEnabledActiveButtons(true);
        }
```

```
        @Override
        public void turnIsSkipped(PlayerActionEvent e) {
            setEnabledButtons(false);
            setDefault();
            drawLettersOnField();
            getNewActiveButtons();
        }

        @Override
        public void letterOnFieldIsChosen(PlayerActionEvent e) {
            setEnabledButtons(false);
            setEnabledCellsAdjacentToLetters(e.letter().cell());
        }

        @Override
        public void turnIsOver(PlayerActionEvent e) {
            setEnabledButtons(false);
            setDefault();
            getNewActiveButtons();
        }

        @Override
        public void cancel(PlayerActionEvent e) {
            setEnabledButtons(false);
            setDefault();
            drawLettersOnField();
        }
    }

    private class GameObserver implements GameListener{

        @Override
        public void gameFinished(GameEvent e) {
            setDefault();
            setEnabledButtons(false);
        }

        @Override
        public void currentLetterIsChosen(GameEvent e) {

        }

        @Override
        public void dictionaryHasNotContainsWord(GameEvent e) {

        }

        @Override
        public void wordHasBeenComposed(GameEvent e) {

        }
    }

    private class MenuObserver implements MenuListener {
        @Override
        public void newGameStarted() {
```

```
                rebuildField();
                setEnabledButtons(false);
                getNewActiveButtons();
            }
        }
    }


public class PlayerScoreWidget extends JPanel {
    private int firstPlayerScore = 0;

    private int secondPlayerScore = 0;

    private JLabel firstPlayerLabel = new JLabel();

    private JLabel secondPlayerLabel = new JLabel();

    public PlayerScoreWidget(GameModel model, GamePanel panel){
        model.addGameListener(new GameObserver());
        model.addPlayerActionListener(new PlayerObserver());
        panel.addMenuListener(new MenuObserver());
    }

    public void buildPlayerScorePane(){
        setLayout(new GridLayout(1, 2));
        setBorder(BorderFactory.createEmptyBorder(5,0,0,0));

        Dimension dimension = new Dimension(300, 50);

        setPreferredSize(dimension);
        setMinimumSize(dimension);
        setMaximumSize(dimension);

        firstPlayerLabel.setText("Счет первого игрока: " +
firstPlayerScore);
        secondPlayerLabel.setText("Счёт второго игрока: " +
secondPlayerScore);

        add(firstPlayerLabel);
        add(secondPlayerLabel);

        setVisible(true);
    }

    private class PlayerObserver implements PlayerActionListener {
        @Override
        public void letterIsPlaced(PlayerActionEvent e) {}
        @Override
        public void letterIsReceived(PlayerActionEvent e) {}
        @Override
        public void turnIsSkipped(PlayerActionEvent e) {}
        @Override
        public void letterOnFieldIsChosen(PlayerActionEvent e) {}
        @Override
        public void turnIsOver(PlayerActionEvent e) {
            if (e.currentWord() != null &&
```

```
!e.currentWord().isEmpty()) {
                if (e.player().name().equals("1")) {
                    firstPlayerScore += e.currentWord().length();
                    firstPlayerLabel.setText("Счет первого игрока:
" + firstPlayerScore);
                }
                else if (e.player().name().equals("2")) {
                    secondPlayerScore += e.currentWord().length();
                    secondPlayerLabel.setText("Счёт второго игрока:
" + secondPlayerScore);
                }
            }
        }
        @Override
        public void cancel(PlayerActionEvent e) {}
    }

    private class GameObserver implements GameListener {
        @Override
        public void gameFinished(GameEvent e) {
            firstPlayerScore = 0;
            secondPlayerScore = 0;
        }
        @Override
        public void currentLetterIsChosen(GameEvent e) {}
        @Override
        public void dictionaryHasNotContainsWord(GameEvent e) {}
        @Override
        public void wordHasBeenComposed(GameEvent e) {}
    }

    private class MenuObserver implements MenuListener {
        @Override
        public void newGameStarted() {
            firstPlayerScore = 0;
            secondPlayerScore = 0;
            firstPlayerLabel.setText("Счет первого игрока: " +
firstPlayerScore);
            secondPlayerLabel.setText("Счёт второго игрока: " +
secondPlayerScore);
        }
    }
}


public class WidgetFactory {
    private final Map<Cell, CellWidget> cells = new HashMap<>();


    //Создать виджет клетки
    public CellWidget createCellWidget(Cell cell){
        if(cells.containsKey(cell)) return cells.get(cell);

        CellWidget cellWidget = new CellWidget(cell);
        cells.put(cell, cellWidget);
        return cellWidget;
```

```
    }
}


public class WordListWidget extends JPanel {
    private DefaultListModel<String> firstPlayerList;

    private DefaultListModel<String> secondPlayerList;

    private JList<String> firstPlayerJList;

    private JList<String> secondPlayerJList;

    public WordListWidget(GameModel model, GamePanel panel){
        model.addPlayerActionListener(new PlayerObserver());
        model.addGameListener(new GameObserver());
        panel.addMenuListener(new MenuObserver());
    }

    public void buildPlayerWordsBoard() {
        setLayout(new GridLayout(1, 2));
        setBorder(BorderFactory.createEmptyBorder(0,0,5,0));

        Dimension dimension = new Dimension(300, 250);

        setPreferredSize(dimension);
        setMinimumSize(dimension);
        setMaximumSize(dimension);

        firstPlayerList = new DefaultListModel<>();
        secondPlayerList = new DefaultListModel<>();
        firstPlayerJList = new JList<>(firstPlayerList);
        secondPlayerJList = new JList<>(secondPlayerList);

        JScrollPane scrollPane1 = new
JScrollPane(firstPlayerJList);
        JScrollPane scrollPane2 = new
JScrollPane(secondPlayerJList);

        add(scrollPane1);
        add(scrollPane2);

        setVisible(true);
    }


    private class PlayerObserver implements PlayerActionListener{
        @Override
        public void letterIsPlaced(PlayerActionEvent e) {}
        @Override
        public void letterIsReceived(PlayerActionEvent e) {}
        @Override
        public void turnIsSkipped(PlayerActionEvent e) {}
        @Override
        public void letterOnFieldIsChosen(PlayerActionEvent e) {}
        @Override
```

```
        public void turnIsOver(PlayerActionEvent e) {
            if (e.currentWord() != null &&
!e.currentWord().isEmpty()) {
                if (e.player().name().equals("1"))
                    firstPlayerList.addElement(e.currentWord());
                else if (e.player().name().equals("2"))
                    secondPlayerList.addElement(e.currentWord());
            }
        }
        @Override
        public void cancel(PlayerActionEvent e) {}
    }

    private class GameObserver implements GameListener{
        @Override
        public void gameFinished(GameEvent e) {
            firstPlayerList.clear();
            secondPlayerList.clear();
        }
        @Override
        public void currentLetterIsChosen(GameEvent e) {}
        @Override
        public void dictionaryHasNotContainsWord(GameEvent e) {}
        @Override
        public void wordHasBeenComposed(GameEvent e) {}
    }

    private class MenuObserver implements MenuListener {
        @Override
        public void newGameStarted() {
            firstPlayerList.clear();
            secondPlayerList.clear();
        }
    }
}


public class GamePanel extends JFrame {
    private FieldWidget fieldWidget;

    private AlphabetWidget alphabetWidget;

    private ActionButtonWidget actionButtonWidget;

    private WordListWidget wordListWidget;

    private  PlayerScoreWidget playerScoreWidget;

    private WidgetFactory widgetFactory = new WidgetFactory();

    private JMenuBar menu;
    private final String fileItems[] = new String[]{"New", "Exit"};

    private final String fieldSizes[] = new String[]{"5x5", "6x6",
"7x7", "8x8"};
```

```
    private GameModel model = new GameModel();

    public GamePanel() {
        super();
        this.setTitle("Балда");

        //Представление должно реагировать на изменение состояния
модели
        model.addGameListener(new GameObserver());

        setLayout(new BorderLayout());

        // Меню
        createMenu();
        setJMenuBar(menu);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Создаем игровое поле
        this.fieldWidget = new FieldWidget(model.field(),
widgetFactory, model, this);
        fieldWidget.createField();
        add(fieldWidget, BorderLayout.WEST);

        // Создаем панель, которую затем поместим на основную
панель
        JPanel centerPanel = new JPanel(new GridLayout(2, 1));

        // Создаем виджет, отображающий буквы алфавита
        this.alphabetWidget = new AlphabetWidget(new Alphabet(),
model, this);
        alphabetWidget.buildLetterPanel();

        centerPanel.add(alphabetWidget);

        // Создаем виджет, отображающий кнопки действий
        this.actionButtonWidget = new ActionButtonWidget(model,
this);
        actionButtonWidget.buildActionButtonsPanel();
        centerPanel.add(actionButtonWidget);

        add(centerPanel, BorderLayout.CENTER);

        // Создаем панель, которую затем поместим на основную
панель
        JPanel eastPanel = new JPanel();
        eastPanel.setLayout(new BoxLayout(eastPanel,
BoxLayout.Y_AXIS));

        eastPanel.setBorder(new EmptyBorder(0,10,0,10));

        // Создаем виджет, отображающий текущий счет игроков
        this.playerScoreWidget = new PlayerScoreWidget(model,
this);
        playerScoreWidget.buildPlayerScorePane();
        eastPanel.add(playerScoreWidget);
```

```
        // Создаем виджет, отображающий списки слов, составленные
игроками
        this.wordListWidget = new WordListWidget(model, this);
        wordListWidget.buildPlayerWordsBoard();
        eastPanel.add(wordListWidget);

        add(eastPanel, BorderLayout.EAST);

        pack();
        setResizable(false);
        setVisible(true);

    }

    // ----------------------------- Создаем меню -----------------
-----------------

    private void createMenu() {

        menu = new JMenuBar();
        JMenu fileMenu = new JMenu("File");

        for (int i = 0; i < fileItems.length; i++) {

            JMenuItem item = new JMenuItem(fileItems[i]);
            item.setActionCommand(fileItems[i].toLowerCase());
            item.addActionListener(new NewMenuListener());
            fileMenu.add(item);
        }
        fileMenu.insertSeparator(1);

        menu.add(fileMenu);
    }

    public class NewMenuListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            String command = e.getActionCommand();
            if ("exit".equals(command)) {
                System.exit(0);
            }
            if ("new".equals(command)) {
                if (applySettings()) {
                    model.startGame();
                    fireNewGameStarted();
                }
            }
        }
    }

    private List<MenuListener> listeners = new ArrayList<>();

    public void addMenuListener(MenuListener l){ listeners.add(l);
}
```

```
    public void removeMenuListener(MenuListener l){
listeners.remove(l); }

    private void fireNewGameStarted(){
        for (MenuListener l : listeners){
            l.newGameStarted();
        }
    }

    private boolean applySettings() {
        HashMap<String, Integer> size = new HashMap<>();
        size.put("5x5", 5);
        size.put("6x6", 6);
        size.put("7x7", 7);
        size.put("8x8", 8);
        String in = (String) JOptionPane.showInputDialog(null,
                "Выберите разсер поля :",
                "Размер поля",
                JOptionPane.QUESTION_MESSAGE,
                null, fieldSizes, fieldSizes[0]);
        if (in != null) {
            int heightAndWidth = size.get(in);
            model.field().setSize(heightAndWidth, heightAndWidth);
            return true;
        }
        return false;
    }

    // ------------------------- Реагируем на изменения модели ----
------------------
        private class GameObserver implements GameListener {

            @Override
            public void gameFinished(GameEvent e) {
                if (e.player() != null) {
                    String str = "Победил игрок" +
e.player().name();
                    JOptionPane.showMessageDialog(null, str,
"Победа!", JOptionPane.INFORMATION_MESSAGE);
                } else {
                    String str = "Ничья";
                    JOptionPane.showMessageDialog(null, str,
"Ничья", JOptionPane.INFORMATION_MESSAGE);
                }
            }
            @Override
            public void currentLetterIsChosen(GameEvent e) {}

            @Override
            public void dictionaryHasNotContainsWord(GameEvent e) {
                int result = JOptionPane.showConfirmDialog(null,
                        "Введённого слова нет в словаре.
Добавить?", "Добавление слова",
                        JOptionPane.YES_NO_OPTION);
                switch (result) {
                    case 0:
```

```
Dictionary.addWord(e.player().currentWord());
                        e.player().endTurn();
                        break;
                    case 1, -1:
                        e.player().cancel();
                        break;
                }
            }

            @Override
            public void wordHasBeenComposed(GameEvent e) {
                JOptionPane.showMessageDialog(null, "Введённое вам
слово уже было составлено", "Отмена",
JOptionPane.INFORMATION_MESSAGE);
                e.player().cancel();
            }
        }
    }



public class Scrabble {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(GamePanel::new);
    }
}
```

4 Вторая итерация разработки

4.1 Функциональные требования (сценарии)

1) Главный успешный сценарий — **Победил один из игроков**

1. Один из двух игроков инициирует новую сессию игры
2. Создается singleton класс словарь
3. Игрок, инициализовавший новую игру, устанавливает размер поля
4. Модель создает поле размером NxN клеток
5. Игровая панель создает визуальный алфавит, список составленных игроками слов и активные кнопки, и отображает все перечисленное, включая созданное моделью поле
6. Модель устанавливает в центр поля случайное слово, равное длине поля
7. Модель назначает первого игрока текущим
8. **Делать**
    1. Текущий игрок выбирает букву посредством визуального алфавита
    2. Текущий игрок выбирает свободную клетку, соседнюю занятой
    3. Игрок выбирает последовательность рядом стоящих букв, включая только что поставленную им, чтобы составить слово
    4. Модель проверяет наличие заданного игроком слова в словаре, **если** слово найдено - добавляет слово в таблицу слов, составленных игроком, и увеличивает счет игрока, составившего его на значение равное длине слова, **иначе** предлагает выбор
        - 8.4.1. Добавить слово в словарь:
            - 8.4.1.1. Игра продолжается, начиная с пункта 8.5
        - 8.4.2. Отменить добавления слова
            - 8.4.2.1. Все действия текущего игрока за этот ход обнуляются и ход начинается заново
    5. Обработчик счёта отображает обновленный счет игрока
    6. Словарь игроков отображает, только что добавленное игроком слово

7. Модель считает, что игра не завершена, **если** на поле есть свободные клетки; **иначе** она считает, что игра завершена
8. Система передает ход другому игроку, **если** игра не завершена

Пока игра не завершена

9. Панель определяет победителем игрока, набравшего больше очков, и выводит сообщение

1.1) Альтернативный сценарий — **Ничья**

Сценарий начинает после п.7 сценария 1)

Делать

1. Текущий игрок выбирает букву посредством визуального алфавита
2. Текущий игрок выбирает свободную клетку, соседнюю занятой
3. Игрок выбирает последовательность рядом стоящих букв, включая только что поставленную им, чтобы составить слово
4. Модель проверяет наличие заданного игроком слова в словаре, **если** слово найдено - добавляет слово в таблицу слов, составленных игроком, и увеличивает счет игрока, составившего его на значение равное длине слова, **иначе** предлагает выбор

    4.1. Добавить слово в словарь:

        4.1.1. Игра продолжается, начиная с пункта 8.5

    4.2. Отменить добавления слова

        4.2.1. Все действия текущего игрока за этот ход обнуляются и ход начинается заново

5. Обработчик счёта отображает обновленный счет игрока
6. Словарь игроков отображает, только что добавленное игроком слово
7. Модель считает, что игра не завершена, **если** на поле есть свободные клетки; **иначе** она считает, что игра завершена
8. Система передает ход другому игроку, **если** игра не завершена

Пока игра не завершена (имеются незанятые клетки)

9. Панель определяет, что счёт равный и выводит сообщение

1.2) Альтернативный сценарий — **Победил один из игроков (оба игрока сдались)**

Сценарий начинает в любой итерации п.8 сценария 1)

1. Текущий игрок пропускает ход
2. Система передает право хода другому игроку
3. Текущий игрок пропускает ход
4. Панель определяет победителем игрока, набравшего наибольшее количество очков, и выводит сообщение

1.3) Альтернативный сценарий — **Ничья (оба игрока сдались)**

Сценарий начинает в любой итерации п.8 сценария 1)

1. Текущий игрок пропускает ход
2. Система передает право хода другому игроку
3. Текущий игрок пропускает ход
4. Панель определяет, что счёт равный, и выводит сообщение

1.4) Альтернативный сценарий главного успешного сценария — **Досрочное завершение игры (всей программы)**

- Сценарий начинается в любой точке сценария 1), **когда** любой из двух игроков инициирует завершение игры
- Система завершает свою работу без каких-либо запросов и предупреждений

2) Успешный сценарий - **Текущий игрок последовательно выбирает занятые клетки соседние последней выбранной для составления слова**

Делать

1. Игрок указывает занятую клетку
2. Поле проверяет, содержит ли клетка поставленную игроком букву, **если** выбранная клетка содержит только что поставленную букву, **то** сделать активной кнопку завершения хода, **иначе** оно считает, что ход еще не может быть завершен
3. Поле показывает список доступных для выбора клеток для составления слова

Пока игрок не подтвердит свой выбор клеток

- Сценарий продолжается после п. 8.3 сценария 1)

2.1) Альтернативный сценарий - **Текущий игрок последовательно выбирает незанятые клетки для составления слова, не выбрав ту, в которую поставил букву**

Делать

1. Сценарий начинает после п.2 сценария 2)
2. В любой момент хода, после установки буквы на поле, у игрока будут доступны две опции – отменить ход и пропустить ход. Так, игрок будет вынужден сделать выбрать один из двух вариантов

Пока игрок не подтвердит свой выбор клеток

- Сценарий начинается заново, с пункта 8 сценария 1)

4.2 Словарь предметной области

Модель – система знает о поле. Модель управляет игровым циклом: определяет очередного игрока, определяет окончание. Определяет победителя

Панель – содержит в себе модель и все элементы, влияющие на отображение игрового процесса. Отображает победившего игрока. Спрашивает у игрока размеры игрового поля при инициализации игры

Поле - квадратная область, состоящая из ячеек (пустых или содержащих буквы). Размеры поля задает игрок.

Ячейка - квадратная область поля. Одновременно может содержать в себе только одну букву.

Слово – последовательность букв, имеющая смысл и содержащаяся в словаре.

Словарь – сборник слов, которые можно составить из букв.

Обработчик счета – ведет подсчет очков, набираемых игроками за составление слов на игровом поле.

Словарь игроков – после окончания хода записывает в один из двух списков слов, составленное игроком слово

Визуальный алфавит – прямоугольная область, наполненная кнопками, с доступными буквами. Используется игроком для установки букв в ячейки.

4.3 Структура программы на уровне классов

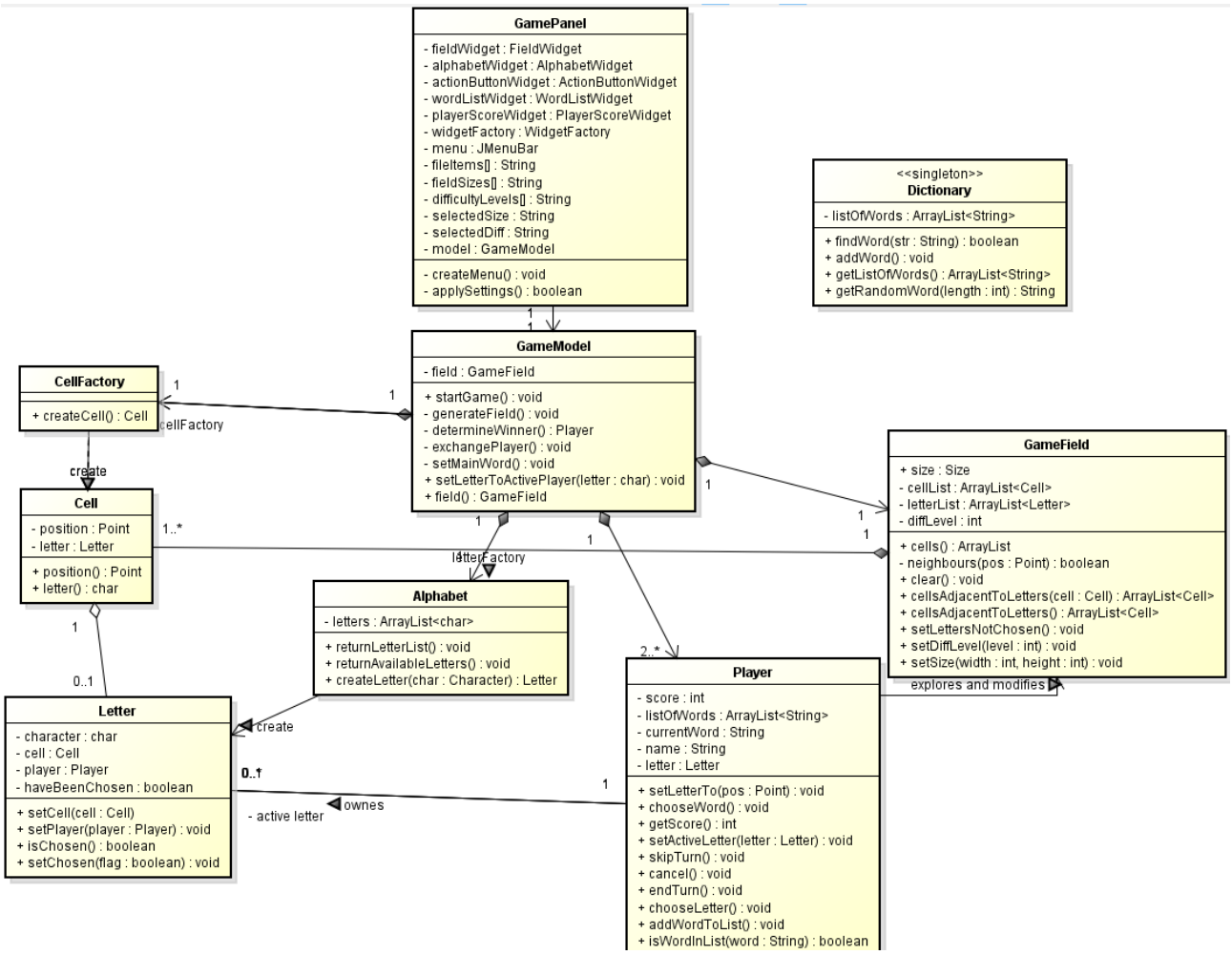


Диаграмма классов вычислительной модели

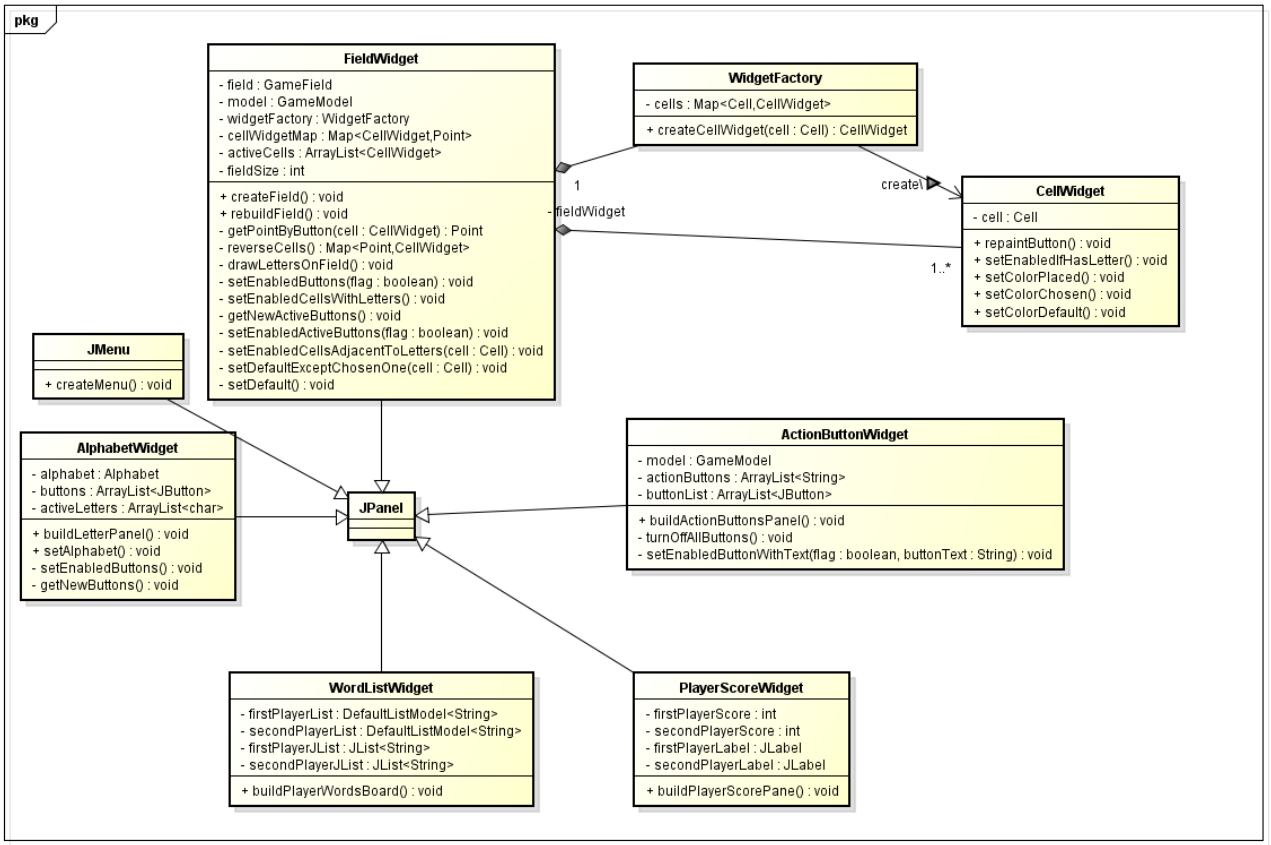
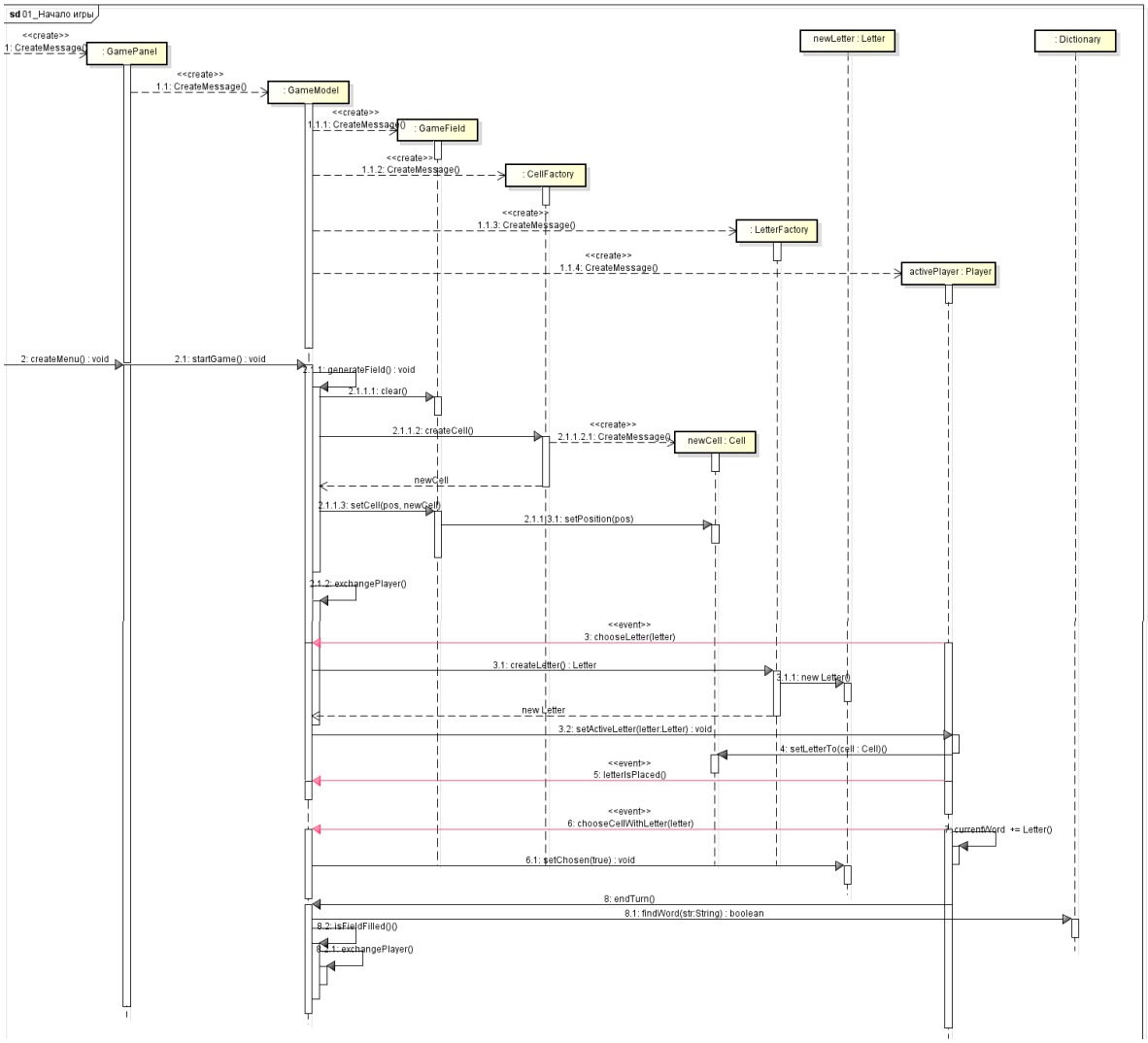
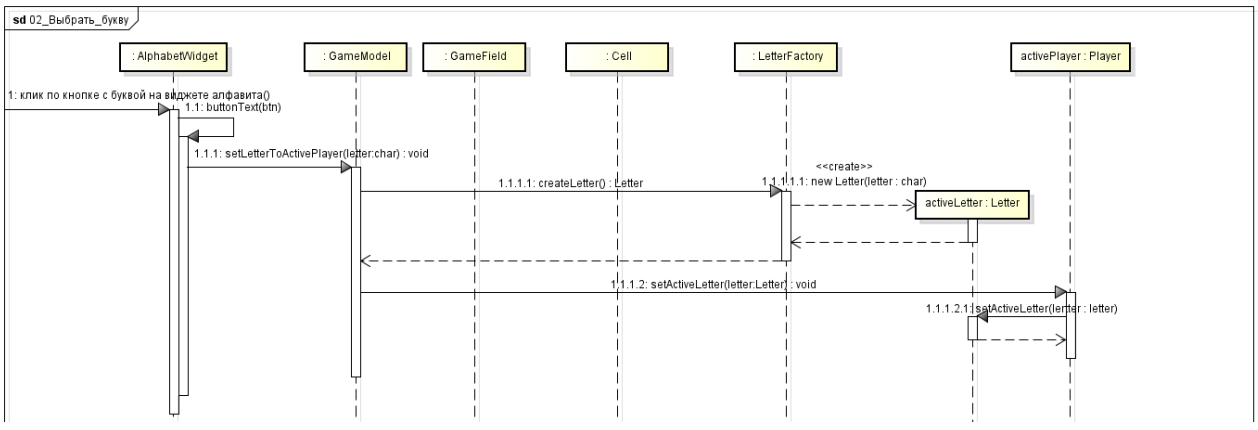


Диаграмма классов представления

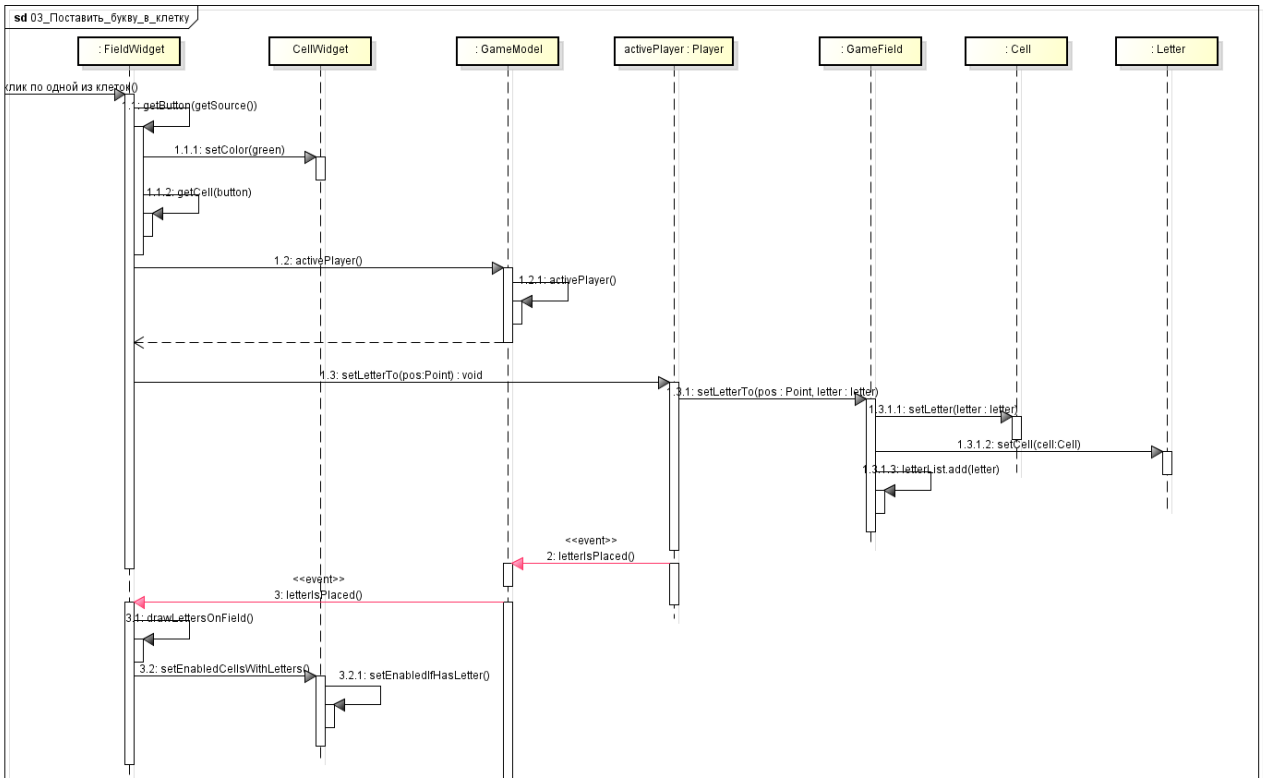
4.4 Типовые процессы в программе



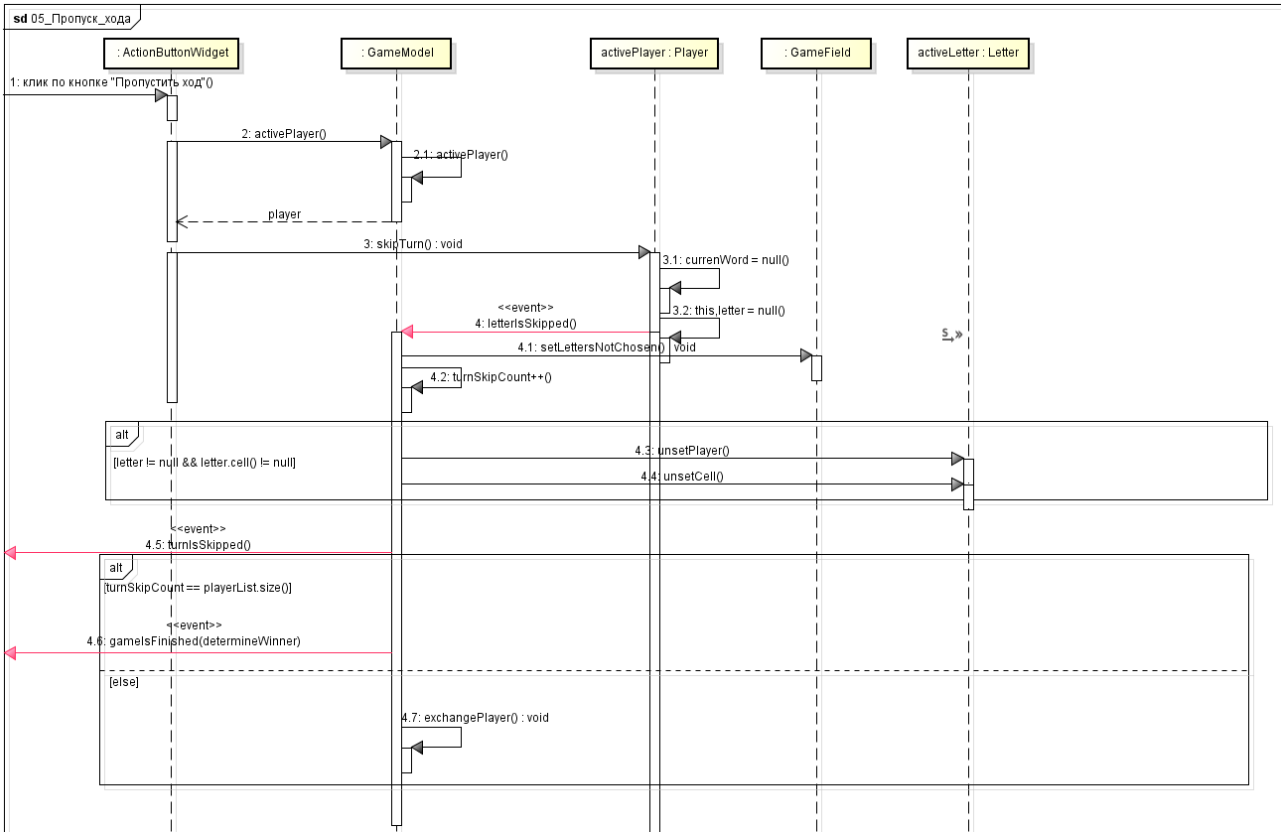
Общий игровой цикл



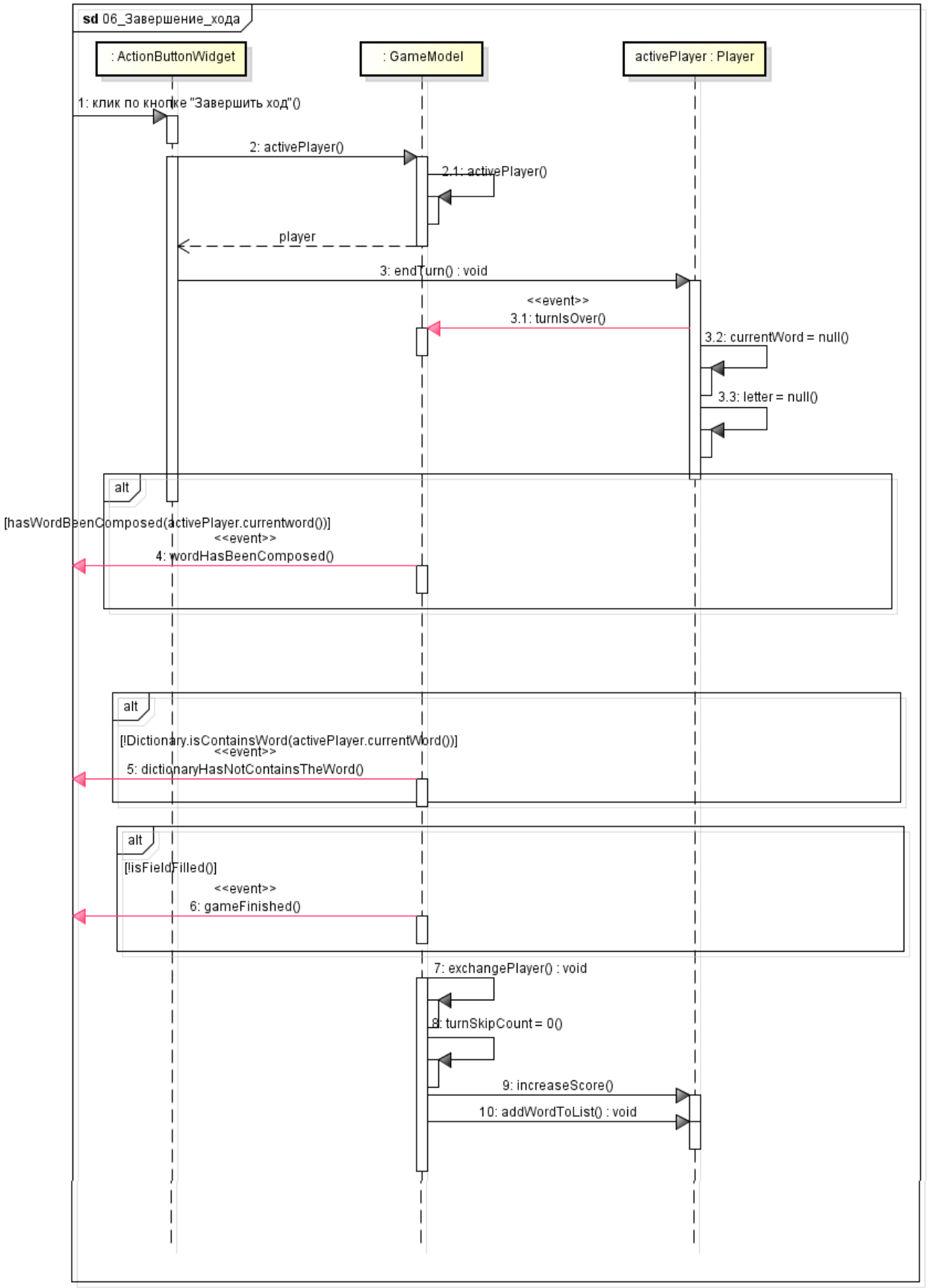
Выбор буквы



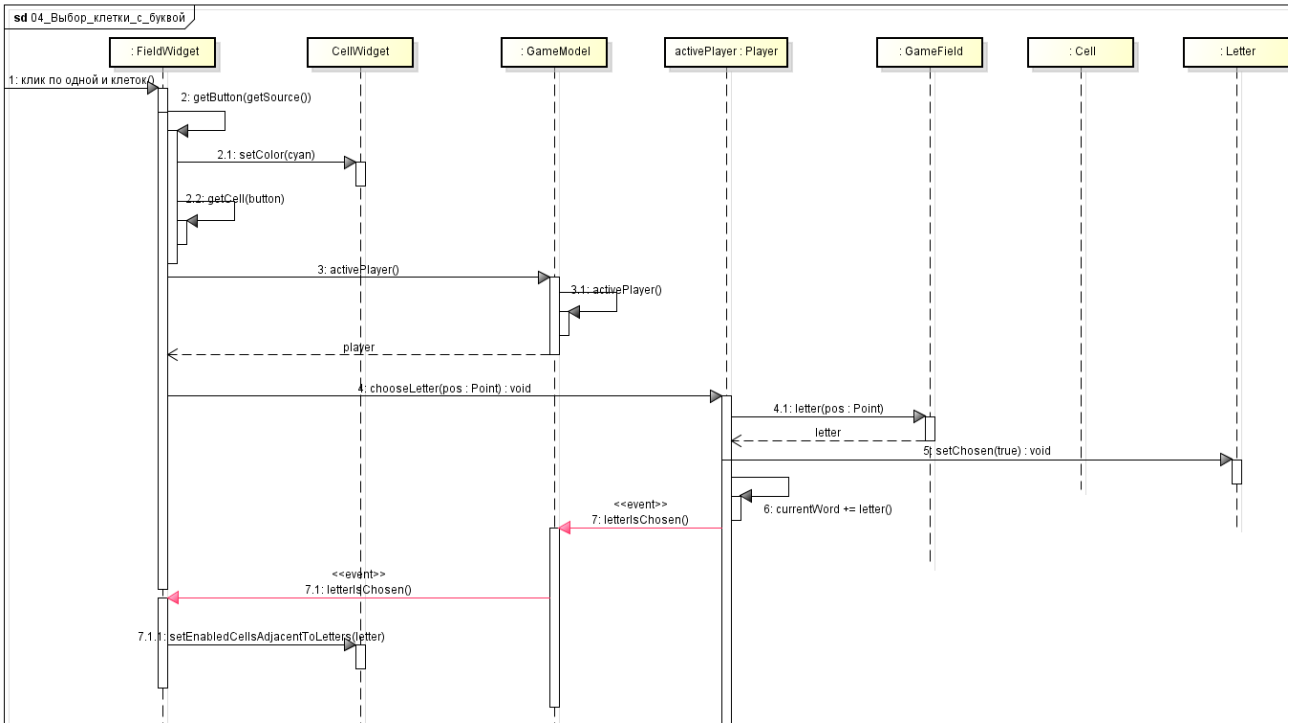
Установка буквы в клетку



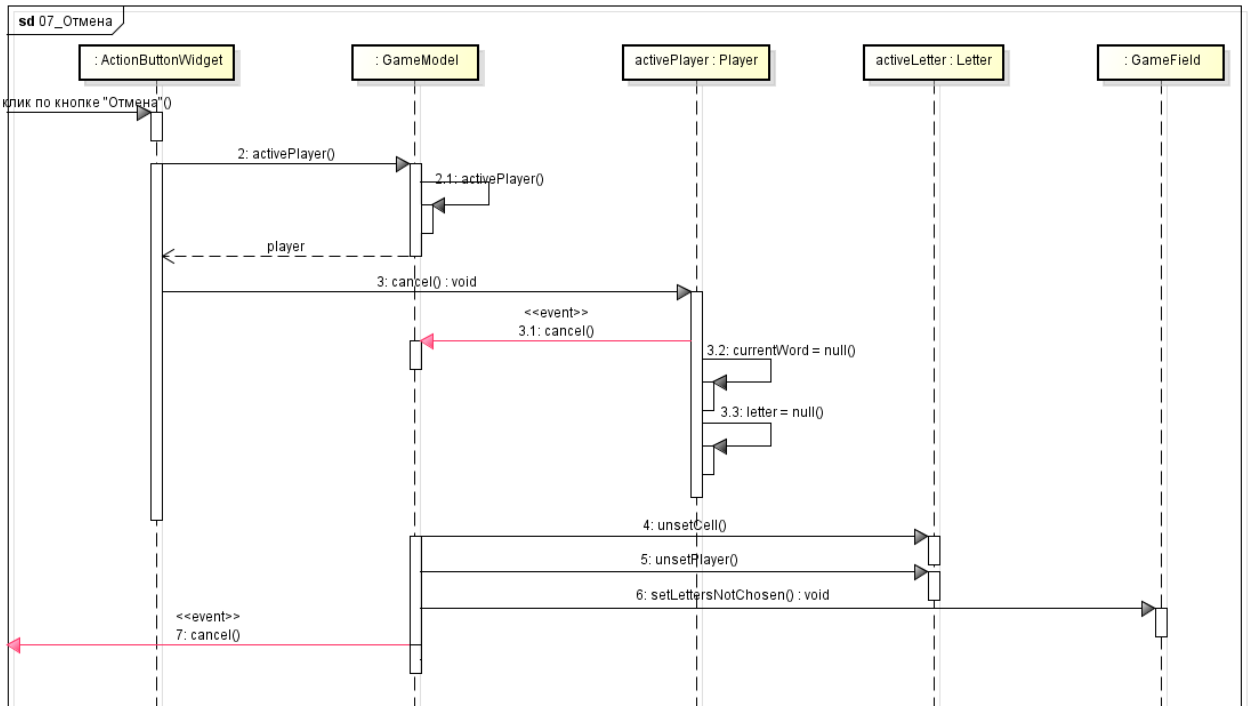
Пропуск хода



Завершение хода



Выбор клетки с буквой



Отмена

4.5 Человеко-машинное взаимодействие

Общий вид главного экрана программы представлен ниже. На нём располагается игровое поле, алфавит доступных букв, кнопки действий, счёт игроков, список слов, составленных игроками.

Всё управление игровым процессом происходит посредствам мыши.

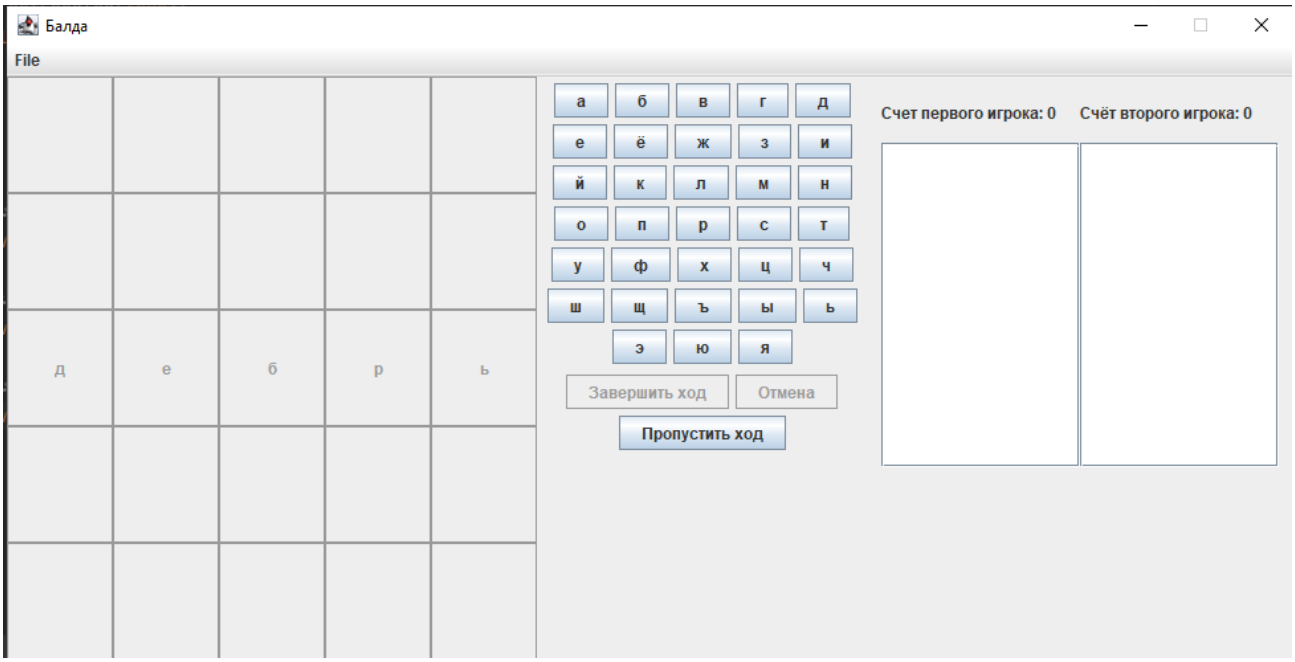


Рис. 1. Общий вид главного экрана программы

При инициализации новой игры появляется меню настроек игрового поля

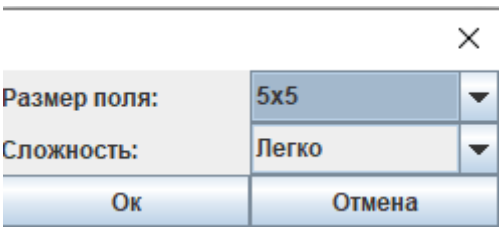


Рис. 2. Меню настроек

В зависимости от выбранных параметров поле и алфавит могут менять свое представление

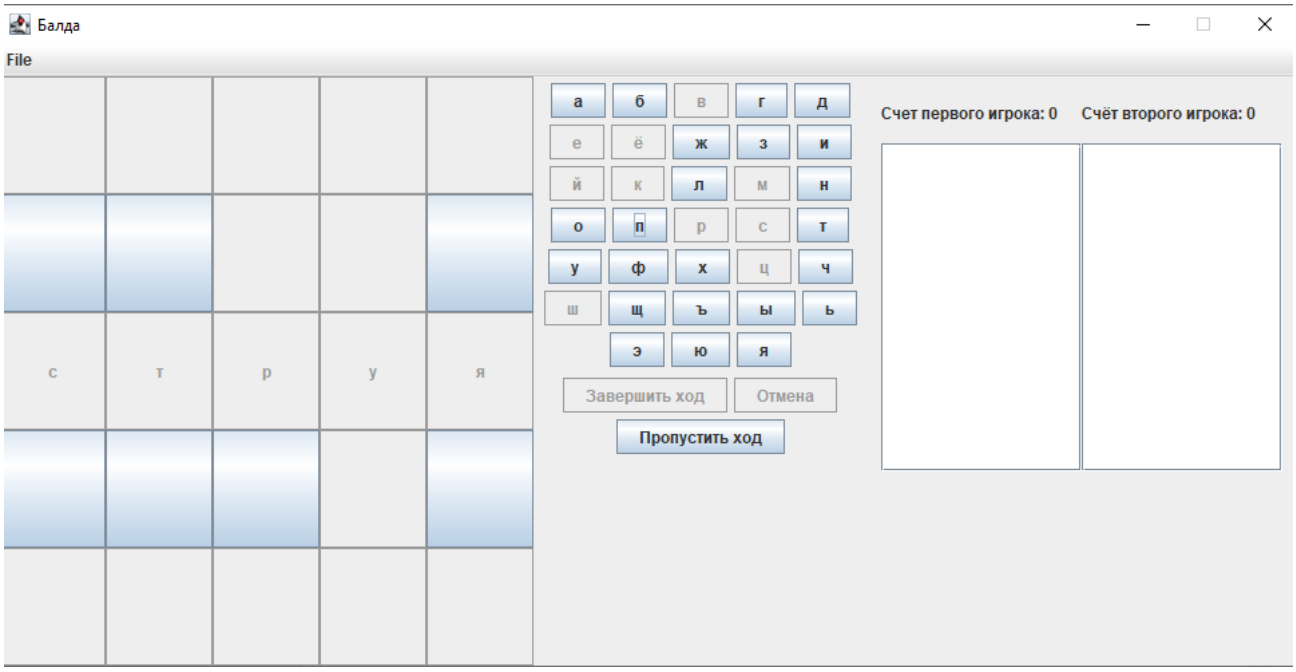


Рис. 3. Вид программы при выборе самого высокого уровня сложности

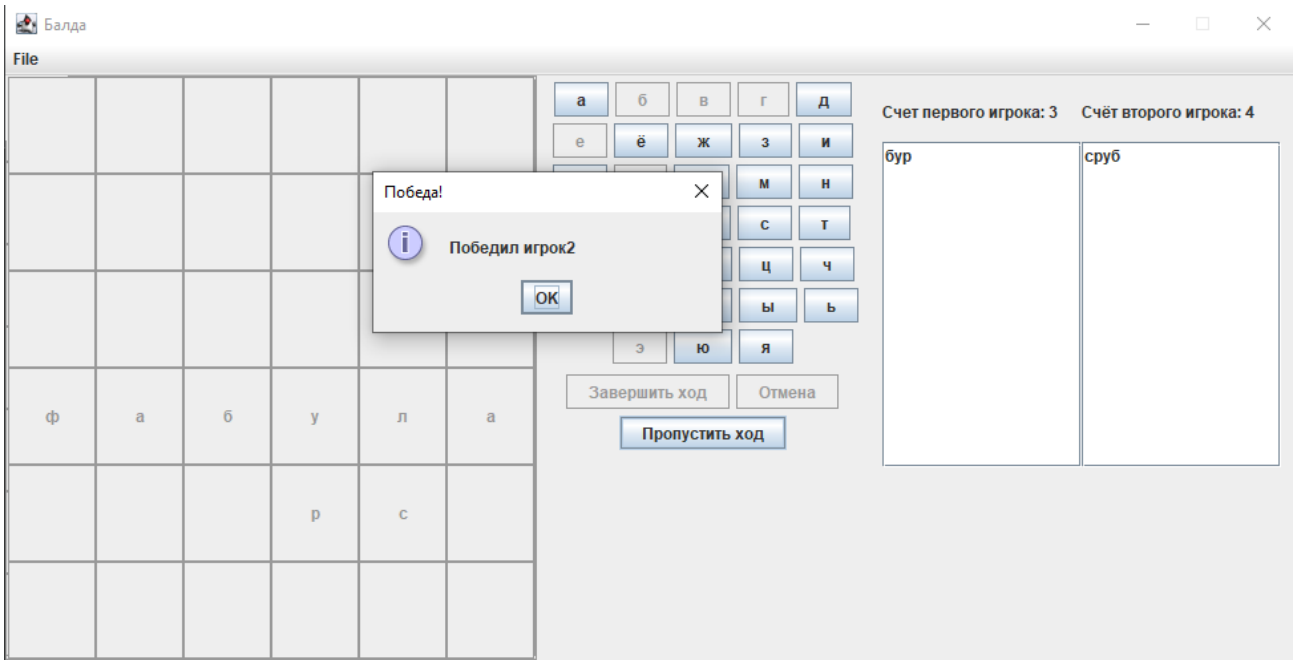


Рис. 4. Пример всплывающего окна победы

4.6 Реализация ключевых классов

```
public class Alphabet extends ComplicatedAlphabet{
    @Override
    public List<Character> returnAvailableLetters(){
        return Collections.unmodifiableList(letters);
    }
}
```

```
public class Cell {
    private final Point coordinates;

//    void setPosition(Point pos){
//        coordinates = pos;
//    }

    public Cell(Point coordinates){
        this.coordinates = coordinates;
    }

    public Point position(){
        return (Point) coordinates.clone();
    }



    // --------- Клетка, принадлежит полю. Принадлежность задает
само поле ------
    private GameField field;

    void setField(GameField f){
        field = f;
    }

    // --------------------- Буква, принадлежащая ячейке ----------
-------------

    private Letter letter;

    public void setLetter(Letter letter) {
        if(letter != null) {
            this.letter = letter;
            letter.setCell(this);
        }else
            this.letter = null;
    }

    public Letter letter(){
        return this.letter;
    }
```

```
    public boolean isAdjacentToLetterCell(){
        Point p = this.position();
        if (this.letter != null){
            return false;
        }
        if(field.cell(new Point(p.x + 1, p.y)) != null &&
field.cell(new Point(p.x + 1, p.y)).letter() != null){
            return true;
        } else if (field.cell(new Point(p.x - 1, p.y)) != null &&
field.cell(new Point(p.x - 1, p.y)).letter() != null) {
            return true;
        }else if (field.cell(new Point(p.x, p.y + 1)) != null &&
field.cell(new Point(p.x, p.y + 1)).letter() != null) {
            return true;
        }else if (field.cell(new Point(p.x, p.y - 1)) != null &&
field.cell(new Point(p.x, p.y - 1)).letter() != null) {
            return true;
        }
        return false;
    }
}


public class ComplicatedAlphabet{
    protected ArrayList<Character> letters = new
ArrayList<>(Arrays.asList(
            'а', 'б', 'в', 'г', 'д',
            'е', 'ё', 'ж', 'з', 'и',
            'й', 'к', 'л', 'м', 'н',
            'о', 'п', 'р', 'с', 'т',
            'у', 'ф', 'х', 'ц', 'ч',
            'ш', 'щ', 'ъ', 'ы', 'ь',
            'э', 'ю', 'я'));

    public List<Character> returnLetterList(){
        return Collections.unmodifiableList(letters);
    }

    public List<Character> returnAvailableLetters(){
        ArrayList<Character> copy = (ArrayList<Character>)
letters.clone();
        Random random = new Random();

        for (int i = 0; i < 10; i++){
            int index = random.nextInt(copy.size());
            copy.remove(index);
        }
        return copy;
    }
}


public class Dictionary {

    private static List<String> listOfWords = new ArrayList<>();
```

```
    private static Dictionary dictionary = new Dictionary();

    private Dictionary() {
        FileReader fr;
        try {
            fr = new
FileReader("\\Users\\Reversi\\java_projects\\buysell\\scrabble\\src
\\resources\\russian.txt");
            Scanner scanner = new Scanner(fr);
            while (scanner.hasNextLine()){
                listOfWords.add(scanner.nextLine());
            }
            fr.close();
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public static Dictionary getDictionary(){
        if(dictionary == null){
            dictionary = new Dictionary();
        }
        return dictionary;
    }

    public static List<String> getListOfWords(){
        return Collections.unmodifiableList(listOfWords);
    }

    public static void addWord(String word){
        FileWriter fw;
        try {
            fw = new
FileWriter("\\Users\\Reversi\\java_projects\\buysell\\scrabble\\src
\\resources\\russian.txt", true);
            fw.write("\n" + word);
            listOfWords.add(word);
            fw.close();
        } catch (FileNotFoundException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }

    public static boolean isContainsWord(String word){
        return listOfWords.contains(word);
    }

    public static String getRandomWord(int len) {
        Random random = new Random();
        String word =
listOfWords.get(random.nextInt(listOfWords.size()));
        while(word.length() != len){
```

```
            word =
listOfWords.get(random.nextInt(listOfWords.size()));
        }
        return word;
    }
}


public class GameField {

    // ------------------------------ Клетки ----------------------
-----------------
    private ArrayList<Cell> cellList = new ArrayList<>();

    public List<Cell> cells(){
        return Collections.unmodifiableList(cellList);
    }

    public Cell cell(Point pos){

        for(Cell obj : cellList)
        {
            if(obj.position().equals(pos))
            { return obj; }
        }

        return null;
    }

    void setCell(Cell cell){
        // Удаляем старую ячейку
        removeCell(cell.position());

        // Связываем ячейку с полем
        cell.setField(this);

        // Добавляем новую ячейку
        cellList.add(cell);
    }

    public void clear(){
        cellList.clear();
    }

    private void removeCell(Point pos){

        Cell obj = cell(pos);
        if(obj != null)     {
            cellList.remove(obj);
            obj.setField(null);
        }
    }

    //Возвращает все клетки, рядом с которыми есть клетка с буквой
    public List<Cell> cellsAdjacentToLetters(){
        ArrayList<Cell> cells = new ArrayList<>();
```

```
        for(Cell c : cellList){
            if(c.isAdjacentToLetterCell())
                cells.add(c);
        }

        if (diffLevel == 2 && cells.size() > 2){
            int numberOfElementsToRemove = cells.size() / 3;

            Random random = new Random();

            for (int i = 0; i < numberOfElementsToRemove; i++){
                cells.remove(random.nextInt(cells.size()));
            }
        }
        return Collections.unmodifiableList(cells);
    }

    private List<Cell> neighbours(Cell cell){
        ArrayList<Cell> cells = new ArrayList<>();
        Point p = cell.position();
        if(cell(new Point(p.x + 1, p.y)) != null) {
                cells.add(cell(new Point(p.x + 1, p.y)));
        }if(cell(new Point(p.x - 1, p.y)) != null) {
                cells.add(cell(new Point(p.x - 1, p.y)));
        }if(cell(new Point(p.x, p.y + 1)) != null) {
                cells.add(cell(new Point(p.x, p.y + 1)));
        }if(cell(new Point(p.x, p.y - 1)) != null) {
                cells.add(cell(new Point(p.x, p.y - 1)));
        }
        return cells;
    }

    //Возвращает клетки-соседи с переданной, в которых есть буква,
и которые еще не были выбраны
    public List<Cell> cellsAdjacentToLetters(Cell cell){
        ArrayList<Cell> cells = new ArrayList<>();
        for(Cell c : neighbours(cell)){
            if(c.letter() != null && !c.letter().isChosen()){
                cells.add(c);
            }
        }

        return Collections.unmodifiableList(cells);
    }

    public void setLettersNotChosen(){
        for (Letter l : letterList){
            l.setChosen(false);
        }
    }

    // ------------------------------ Буквы -----------------------
----------------

    private ArrayList<Letter> letterList = new ArrayList<>();
```

```
    public List<Letter> letters() {
        letterList.clear();
        for (Cell c : cellList){
            Letter l = c.letter();
            if (l != null)
            letterList.add(c.letter());
        }
        return Collections.unmodifiableList(letterList);
    }

    public Letter letter(Point pos){
        Cell obj = cell(pos);
        if(obj != null) return obj.letter();
        return null;
    }

    public void setLetter(Point pos, Letter letter){
        Cell obj = cell(pos);
        if(obj != null){
            obj.setLetter(letter);
            letter.setCell(obj);
            letterList.add(letter);
        }
    }

    private int diffLevel = 2;

    public void setDiffLevel(int level) {
        if(level != 1 && level != 2) throw new RuntimeException("B
поле передан неправильный уровень сложности!");
        diffLevel = level;
    }

    // ----------------------- Ширина и высота поля ---------------
---------------

    public GameField(){
        setSize(5, 5);
    }
    private int width;
    private int height;


    public void setSize(int width, int height) {

        this.width = width;
        this.height = height;

        // Удаляем все ячейки находящиеся вне поля
        ArrayList<Cell> removableList = new ArrayList<>();
        for (Cell obj : cellList) {
            if(!containsRange(obj.position())) {
                removableList.add(obj);
            }
        }
        for (Cell c : removableList) {
```

```
                cellList.remove(c);
            }
        }

        public int width() {
            return width;
        }

        public int height() {
            return height;
        }

        public boolean containsRange(Point p){
            return p.getX() >= 0 && p.getX() < width &&
                    p.getY() >= 0 && p.getY() < height;
        }
    }


    public class GameModel {

        // ------------------------------- Поле ----------------------
    ---------------
        private GameField field = new GameField();

        public GameField field() {
            return this.field;
        }

        // ------------------------------- Игроки --------------------
    ---------------
        private ArrayList<Player> playerList = new ArrayList<>();
        private int activePlayer;

        private int turnSkipCount = 0;
        public Player activePlayer(){
            return playerList.get(activePlayer);
        }

        public GameModel(){
            //Размеры поля по умолчанию
            field.setSize(8, 8);

            //Создание игроков
            Player p;
            PlayerObserver observer = new PlayerObserver();

            p = new Player(field(), "1", this);
            p.addPlayerActionListener(observer);
            playerList.add(p);
            activePlayer = 0;

            p = new Player(field(), "2", this);
            p.addPlayerActionListener(observer);
            playerList.add(p);
        }
```

```
    // ---------------------- Порождение обстановки на поле -------
-------------

    private CellFactory cellFactory = new CellFactory();

    private void generateField(){

        field().clear();
        for(int row = 0; row < this.field().height(); row++) {
            for(int col = 0; col < this.field().width(); col++) {
                field().setCell(cellFactory.createCell(new
Point(col, row)));
            }
        }
    }

    // ----------------------------- Игровой процесс --------------
-------------

    public void startGame(){
        //Генерируем поле
        generateField();
        setMainWord();

        //Передаем ход первому игроку
        activePlayer = playerList.size()-1;
        exchangePlayer();
    }

    private LetterFactory letterFactory = new LetterFactory();
    public void exchangePlayer(){
        activePlayer++;
        if(activePlayer >= playerList.size()) activePlayer = 0;

        field.setLettersNotChosen();

        if(isFieldFilled()){
            fireGameFinished(determineWinner());
        }
    }

    private void setMainWord() {
        String word = Dictionary.getRandomWord(field().width());
        for (int i = 0; i < field().width(); i++){
            field().setLetter(new Point(i, field().height()/2),
letterFactory.createLetter(word.charAt(i)));
        }
    }
    public void setLetterToActivePlayer(char ch){

activePlayer().setActiveLetter(letterFactory.createLetter(ch));
    }

    private boolean isFieldFilled(){
        for (Cell cell : field.cells()){
```

```
                if(cell.letter() == null){
                    return false;
                }
            }
            return true;
        }

        private boolean hasWordBeenComposed(String word){
            for (Player p : playerList){
                boolean flag = p.isWordInList(word);
                if(flag == true)
                    return true;
            }
            return false;
        }

        public Player determineWinner(){
            turnSkipCount = 0;
            if (playerList.get(0).score() > playerList.get(1).score()){
                return playerList.get(0);
            }
            else if (playerList.get(0).score() <
    playerList.get(1).score()) {
                return playerList.get(1);
            }
            else return null;
        }

        // ------------------------- Реагируем на действия игрока -----
    -------------

        private class PlayerObserver implements PlayerActionListener {

            @Override
            public void letterIsPlaced(PlayerActionEvent e) {
                if (e.player() == activePlayer()) {
                    fireLetterIsPlaced(e);
                }
            }

            @Override
            public void letterIsReceived(PlayerActionEvent e) {
                if (e.player() == activePlayer()) {
                    fireLetterIsReceived(e);
                }
            }

            @Override
            public void turnIsSkipped(PlayerActionEvent e) {
                if (e.letter() != null) {
                    if (e.letter().cell() != null) {
                        Letter l = e.letter();
                        l.unsetCell();
                        l.unsetPlayer();
                    }
                }
```

```
            if (e.player() == activePlayer()) {
                fireTurnIsSkipped(e);
            }

            turnSkipCount++;
            field.setLettersNotChosen();

            if (turnSkipCount == playerList.size()) {
                fireGameFinished(determineWinner());
            } else {
                exchangePlayer();
            }
        }

        @Override
        public void letterOnFieldIsChosen(PlayerActionEvent e) {
            if (e.player() == activePlayer()) {
                fireLetterOnFieldIsChosen(e);
            }

            if (e.letter() == e.player().activeLetter()) {
                fireCurrentLetterIsChosen(e.player());
            }
        }

        @Override
        public void turnIsOver(PlayerActionEvent e) {
            if (hasWordBeenComposed(e.player().currentWord())) {
                fireWordHasBeenComposed(activePlayer());
                return;
            }
            if
(!Dictionary.isContainsWord(e.player().currentWord())) {
                fireDictionaryHasNotContainsWord(e.player());
                return;
            }

            if (e.player() == activePlayer()) {
                fireTurnIsOver(e);
            }

            if (!isFieldFilled()) {
                exchangePlayer();
                turnSkipCount = 0;
                e.player().increaseScore(e.currentWord().length());
                e.player().addWordToList();
                System.out.println("Игрок: " + e.player().name() +
" добавил слово " + e.player().currentWord() + " Счёт игрока:" +
e.player().score());

            } else {
                fireGameFinished(determineWinner());
            }
        }

        @Override
```

```
        public void cancel(PlayerActionEvent e) {
            Letter l = e.letter();
            l.unsetCell();
            l.unsetPlayer();
            fireCancel(e);
            field.setLettersNotChosen();
        }
    }

        // ------------------------ Порождает события игры --------
--------------------

        private ArrayList<GameListener> gameListenerList = new
ArrayList<>();

        public void addGameListener(GameListener l) {
            gameListenerList.add(l);
        }

        public void removeGameListener(GameListener l) {
            gameListenerList.remove(l);
        }

        protected void fireGameFinished(Player winner) {
            GameEvent event = new GameEvent(this);
            event.setPlayer(winner);
            for (GameListener listener : gameListenerList) {
                listener.gameFinished(event);
            }
        }

        protected void fireCurrentLetterIsChosen(Player p) {
            GameEvent event = new GameEvent(this);
            event.setPlayer(p);
            for (GameListener listener : gameListenerList) {
                listener.currentLetterIsChosen(event);
            }
        }

    protected void fireDictionaryHasNotContainsWord(Player p) {
        GameEvent event = new GameEvent(this);
        event.setPlayer(p);
        for (GameListener listener : gameListenerList) {
            listener.dictionaryHasNotContainsWord(event);
        }
    }

    protected void fireWordHasBeenComposed(Player p) {
        GameEvent event = new GameEvent(this);
        event.setPlayer(p);
        for (GameListener listener : gameListenerList) {
            listener.wordHasBeenComposed(event);
        }
    }

        // --------------------- Порождает события, связанные с
```

```
игроками ------------

    private ArrayList<PlayerActionListener>
playerActionListenerList = new ArrayList<>();

    public void addPlayerActionListener(PlayerActionListener l) {
        playerActionListenerList.add(l);
    }

    public void removePlayerActionListener(PlayerActionListener l)
{
        playerActionListenerList.remove(l);
    }

    protected void fireLetterIsPlaced(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.letterIsPlaced(e);
        }
    }

    protected void fireLetterIsReceived(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.letterIsReceived(e);
        }
    }

    protected void fireTurnIsSkipped(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.turnIsSkipped(e);
        }
    }

    protected void fireLetterOnFieldIsChosen(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.letterOnFieldIsChosen(e);
        }
    }

    protected void fireTurnIsOver(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.turnIsOver(e);
        }
    }

    protected void fireCancel(PlayerActionEvent e) {
        for (PlayerActionListener listener :
playerActionListenerList) {
            listener.cancel(e);
        }
    }
}
```

```java
public class Letter {

    // --------------------- Буква -----------------------
    private final char character;

    public Letter(char ch){
        this.character = ch;
    }

    public char character(){
        return this.character;
    }

    // --------------------- Клетка -----------------------
    private Cell cell = null;

    public Cell cell(){ return this.cell; }

    public void setCell(Cell cell) {
        if(this.cell == null){
            this.cell = cell;
            cell.setLetter(this);
        }
    }

    public void unsetCell(){
        this.cell.setLetter(null);
        this.cell = null;
    }

    //Была ли буква выбрана
    private boolean haveBeenChosen = false;

    public boolean isChosen() {
        return haveBeenChosen;
    }

    public void setChosen(boolean haveChosen) {
        this.haveBeenChosen = haveChosen;
    }

    // Игрок, которому прнадлежит буква. Буква может быть
нейтральной (не принадлежать никому) -

    private Player player = null;

    public Player player(){ return this.player; }

    public void setPlayer(Player p){ this.player = p; }

    public void unsetPlayer(){
        this.player = null;
    }
}
```

```
public class Player {
    // ------------------------------- Поля связанные с игроком -
------------------------------
    private int playerScore = 0;

    private String name;

    public String name(){
        return this.name;
    }

    private String currentWord = "";

    public String currentWord(){
        return currentWord;
    }


    // ----------------------- Устанавливаем связь с полем --------
---------------
    GameField field;

    public Player (GameField field, String name, GameModel model) {
        model.addGameListener(new GameObserver());
        this.field = field;
        this.name = name;
    }

    // ---------------------- Счёт игрока -----------------------

    public int score(){
        return playerScore;
    }

    public void increaseScore(int points){
        playerScore += points;
    }

    // ---------------------- Слова, составленные игроком ---------
--------------
    private ArrayList<String> words = new ArrayList<>();

    public boolean isWordInList(String word){
        return words.contains(word);
    }

    public void addWordToList(){
        if(currentWord.length() > 0)
            words.add(currentWord);
    }


    // ---------------------- Буква, которую нужно установить -----
------------------
```

```
    private Letter letter;

    public void setActiveLetter(Letter l) {
        this.letter = l;
        l.setPlayer(this);

        // Генерируем событие
        fireLetterIsReceived(this.letter);
    }

    public Letter activeLetter() {
        return this.letter;
    }

    public void setLetterTo(Point pos){

        if (this.letter == null) throw new
IllegalArgumentException("Буква не была задана!");

        this.field.setLetter(pos, this.letter);

        // Генерируем событие
        if(this.field.letter(pos) == this.letter) {
            fireLetterIsPlaced(this.letter);
            //this.letter = null;
        }
    }

    // ---------------------- Процесс игры -----------------------

    public void chooseLetter(Point pos){
        Letter chosenLetter = field.letter(pos);
        if(chosenLetter == null) throw new FindException("В
выбранной клетке нет буквы!");

        chosenLetter.setChosen(true);
        currentWord += String.valueOf(chosenLetter.character());

        fireLetterOnFieldIsChosen(chosenLetter);
    }

    public void skipTurn(){
        fireTurnIsSkipped(this.letter);
        currentWord = "";
        this.letter = null;
    }

    public void endTurn(){
        fireTurnIsOver(this.letter, this.currentWord);
        currentWord = "";
        this.letter = null;
    }

    public void cancel(){
        fireCancel(this.letter, this.currentWord);
        currentWord = "";
```

```
        this.letter = null;
    }

    // ---------------------- Порождает события -------------------
----------

   private ArrayList<PlayerActionListener> listeners = new
ArrayList<>();

    // Присоединяет слушателя
    public void addPlayerActionListener(PlayerActionListener l) {
        this.listeners.add(l);
    }

    // Отсоединяет слушателя
    public void removePlayerActionListener(PlayerActionListener l)
{
        this.listeners.remove(l);
    }

    // Оповещает слушателей о событии
    protected void fireLetterIsPlaced(Letter l) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        for(PlayerActionListener o : listeners){
            o.letterIsPlaced(event);
        }
    }

    protected void fireLetterIsReceived(Letter l) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        for(PlayerActionListener o : listeners){
            o.letterIsReceived(event);
        }
    }

    protected void fireTurnIsSkipped(Letter l) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        for (PlayerActionListener o : listeners){
            o.turnIsSkipped(event);
        }
    }

    protected void fireLetterOnFieldIsChosen(Letter l) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        for (PlayerActionListener o : listeners){
            o.letterOnFieldIsChosen(event);
        }
    }
```

```
    protected void fireTurnIsOver(Letter l, String word) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        event.setCurrentWord(word);
        for (PlayerActionListener o : listeners){
            o.turnIsOver(event);
        }
    }

    protected void fireCancel(Letter l, String word) {
        PlayerActionEvent event = new PlayerActionEvent(this);
        event.setPlayer(this);
        event.setLetter(l);
        event.setCurrentWord(word);
        for (PlayerActionListener o : listeners){
            o.cancel(event);
        }
    }

    private class GameObserver implements GameListener{

        @Override
        public void gameFinished(GameEvent e) {
            playerScore = 0;
        }
        @Override
        public void currentLetterIsChosen(GameEvent e) {}
        @Override
        public void dictionaryHasNotContainsWord(GameEvent e) {}
        @Override
        public void wordHasBeenComposed(GameEvent e) {}
    }
}


public class GameEvent extends EventObject {

    Player player;

    public void setPlayer(Player player) {
        this.player = player;
    }

    public Player player() {
        return player;
    }

    public GameEvent(Object source){
        super(source);
    }


public interface GameListener extends EventListener {
    public void gameFinished(GameEvent e);
```

```
    public void currentLetterIsChosen(GameEvent e);

    public void dictionaryHasNotContainsWord(GameEvent e);

    public void wordHasBeenComposed(GameEvent e);
}



public interface MenuListener extends EventListener {
    public void newGameStarted();
}


public class PlayerActionEvent extends EventObject {
    // ------------------------------- Игрок --------------------
-----------------
    Player player;

    public void setPlayer(Player player) {
        this.player = player;
    }

    public Player player() {
        return player;
    }

    // ------------------------------- Активная буква ------------
-------------------------
    Letter letter;

    public void setLetter(Letter letter) {
        this.letter = letter;
    }

    public Letter letter() {
        return letter;
    }

    // ------------------------------- Составленное слово --------
-----------------------------
    String currentWord;

    public void setCurrentWord(String currentWord) {
        this.currentWord = currentWord;
    }

    public String currentWord() {
        return currentWord;
    }

    public PlayerActionEvent(Object source) {
        super(source);
```

```
    }
}


public interface PlayerActionListener extends EventListener {
    void letterIsPlaced(PlayerActionEvent e);

    void letterIsReceived(PlayerActionEvent e);

    void turnIsSkipped(PlayerActionEvent e);

    void letterOnFieldIsChosen(PlayerActionEvent e);

    void turnIsOver(PlayerActionEvent e);

    void cancel(PlayerActionEvent e);
}


public class CellFactory {
    public Cell createCell(Point point){
        return new Cell(point);
    }
}


public class LetterFactory {
    public Letter createLetter(char ch){
        return new Letter(ch);
    }
}


public class ActionButtonWidget extends JPanel {
    private GameModel model;

    private List<String> actionButtons = Arrays.asList("Завершить
ход", "Отмена", "Пропустить ход");

    private List<JButton> buttonList = new ArrayList<>();

    public ActionButtonWidget(GameModel model, GamePanel panel){
        model.addPlayerActionListener(new PlayerObserver());
        model.addGameListener(new GameObserver());
        panel.addMenuListener(new MenuObserver());
        this.model = model;
    }

    public void buildActionButtonsPanel(){
        setLayout(new FlowLayout());
        Dimension dimension = new Dimension(250, 40);

        setPreferredSize(dimension);

        JButton btn;
```

```
        for (String str : actionButtons){
            btn = new JButton(str);
            add(btn);
            btn.addActionListener(new ActionButtonClickListener());
            buttonList.add(btn);
            btn.setEnabled(false);
        }
    }

    private void turnOffAllButtons(){
        for (JButton button : buttonList){
            button.setEnabled(false);
        }
    }

    private void setEnabledButtonWithText(boolean flag, String
buttonText){
        for (JButton button : buttonList){
            if(button.getText().equals(buttonText)){
                button.setEnabled(flag);
            }
        }
    }

    private class ActionButtonClickListener implements
ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            JButton btn = (JButton) e.getSource();
            String btnText = btn.getText();
            System.out.println(btnText);
            if (btnText.equals("Пропустить ход")) {
                model.activePlayer().skipTurn();
            } else if (btnText.equals("Завершить ход")) {
                model.activePlayer().endTurn();
            } else if (btnText.equals("Отмена")) {
                model.activePlayer().cancel();
            }
        }
    }

    private class PlayerObserver implements PlayerActionListener{

        @Override
        public void letterIsPlaced(PlayerActionEvent e) {
            setEnabledButtonWithText(true, "Отмена");
        }

        @Override
        public void letterIsReceived(PlayerActionEvent e) {

        }

        @Override
        public void turnIsSkipped(PlayerActionEvent e) {
            setEnabledButtonWithText(false, "Завершить ход");
```

```
                setEnabledButtonWithText(false, "Отмена");
            }

            @Override
            public void letterOnFieldIsChosen(PlayerActionEvent e) {

            }

            @Override
            public void turnIsOver(PlayerActionEvent e) {
                setEnabledButtonWithText(false, "Завершить ход");
                setEnabledButtonWithText(false, "Отмена");
            }

            @Override
            public void cancel(PlayerActionEvent e) {
                setEnabledButtonWithText(false, "Завершить ход");
                setEnabledButtonWithText(false, "Отмена");
            }
        }

        private class GameObserver implements GameListener{

            @Override
            public void gameFinished(GameEvent e) {
                turnOffAllButtons();
            }

            @Override
            public void currentLetterIsChosen(GameEvent e) {
                setEnabledButtonWithText(true, "Завершить ход");
            }

            @Override
            public void dictionaryHasNotContainsWord(GameEvent e) {

            }

            @Override
            public void wordHasBeenComposed(GameEvent e) {

            }
        }

        private class MenuObserver implements MenuListener {
            @Override
            public void newGameStarted() {
                turnOffAllButtons();
                setEnabledButtonWithText(true, "Пропустить ход");
            }
        }
    }


    public class AlphabetWidget extends JPanel {
        private GameModel model;
```

```
    private ComplicatedAlphabet alphabet;

    private ArrayList<JButton> buttonList = new ArrayList<>();

    private List<Character> activeLetters;

    public AlphabetWidget(ComplicatedAlphabet alphabet, GameModel
model, GamePanel panel){
        model.addGameListener(new GameObserver());
        model.addPlayerActionListener(new PlayerObserver());
        panel.addMenuListener(new MenuObserver());
        this.model = model;
        this.alphabet = alphabet;
    }

    public void buildLetterPanel(){
        List<Character> letters;
        letters = alphabet.returnLetterList();

        setLayout(new FlowLayout());
        Dimension dimension = new Dimension(250, 220);
        setPreferredSize(dimension);
        JButton btn;
        for (Character letter : letters){
            btn = new JButton(Character.toString(letter));
            add(btn);
            btn.addActionListener(new LetterClickListener());
            buttonList.add(btn);
            btn.setEnabled(false);
        }
        activeLetters = alphabet.returnAvailableLetters();
    }

    public void setAlphabet(ComplicatedAlphabet alphabet){
        this.alphabet = alphabet;
    }

    private void setEnabledButtons(boolean flag){
        for (Character ch : activeLetters){
            for (JButton button : buttonList){
                if(button.getText().equals(ch.toString()))
                    button.setEnabled(flag);
            }
        }
    }

    private void setEnabledAllButtons(boolean flag){
        for (JButton button : buttonList){
            button.setEnabled(flag);
        }
    }

    private void getNewButtons(){
        activeLetters = alphabet.returnAvailableLetters();
    }
```

```
private class LetterClickListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        JButton btn = (JButton) e.getSource();
        String btnText = btn.getText();
        System.out.println(btnText);
        model.setLetterToActivePlayer(btnText.charAt(0));
    }
}

private class PlayerObserver implements PlayerActionListener{
    @Override
    public void letterIsPlaced(PlayerActionEvent e) {
        setEnabledAllButtons(false);
    }
    @Override
    public void letterIsReceived(PlayerActionEvent e) {}

    @Override
    public void turnIsSkipped(PlayerActionEvent e) {
        setEnabledAllButtons(false);
        getNewButtons();
        setEnabledButtons(true);
    }
    @Override
    public void letterOnFieldIsChosen(PlayerActionEvent e) {}

    @Override
    public void turnIsOver(PlayerActionEvent e) {
        setEnabledAllButtons(false);
        getNewButtons();
        setEnabledButtons(true);
    }

    @Override
    public void cancel(PlayerActionEvent e) {
        setEnabledButtons(true);
    }
}

private class GameObserver implements GameListener{
    @Override
    public void gameFinished(GameEvent e) {
        setEnabledAllButtons(false);
    }
    @Override
    public void currentLetterIsChosen(GameEvent e) {}
    @Override
    public void dictionaryHasNotContainsWord(GameEvent e) {}
    @Override
    public void wordHasBeenComposed(GameEvent e) {}
}

private class MenuObserver implements MenuListener{
    @Override
```

```
        public void newGameStarted() {
            setEnabledAllButtons(false);
            getNewButtons();
            setEnabledButtons(true);
        }
    }
}


public class CellWidget extends JButton {

    private Cell cell;


    public CellWidget(Cell cell){
        if(cell == null) throw new RuntimeException("В CellWidget
передан null");
        else{
            this.cell = cell;
            if(cell.letter() != null)
                setText(String.valueOf(cell.letter().character()));
            //setBackground(Color.WHITE);

            setFocusable(false);
            setVisible(true);
        }
    }

    public void repaintButton(){
        if(cell.letter() != null){
            setText(String.valueOf(cell.letter().character()));
        } else {
            setText(null);
        }
        revalidate();
        repaint();
    }

    public void setEnabledIfHasLetter(){
        if (cell.letter() != null){
            setEnabled(true);
        } else {
            //setColorDefault();
            setEnabled(false);
        }
    }

    public void setColorPlaced(){
        setBackground(Color.GREEN);
    }

    public void setColorChosen(){
        setBackground(Color.CYAN);
    }

    public void setColorDefault(){
```

```
            setBackground(null);
        }
    }


    public class FieldWidget extends JPanel {
        private GameModel model;

        private GameField field;

        private WidgetFactory factory = new WidgetFactory();

        private final Map<CellWidget, Point> cellWidgetMap = new
    HashMap<>();

        private ArrayList<CellWidget> activeCells = new ArrayList<>();

        private final int FIELD_SIZE = 400;

        public FieldWidget(GameField field, GameModel model, GamePanel
    panel) {
            model.addPlayerActionListener(new PlayerObserver());
            model.addGameListener(new GameObserver());
            panel.addMenuListener(new MenuObserver());
            this.model = model;
            this.field = field;
        }

        public void createField() {
            setLayout(new GridLayout(field.height(), field.width()));

            Dimension dimension = new Dimension(FIELD_SIZE,
    FIELD_SIZE);

            setPreferredSize(dimension);
            setMinimumSize(dimension);
            setMaximumSize(dimension);

            setVisible(true);
        }

        public void rebuildField() {
            removeAll();
            cellWidgetMap.clear();

            setLayout(new GridLayout(field.height(), field.height()));

            for (int row = 0; row < field.height(); row++) {
                for (int col = 0; col < field.width(); col++) {
                    Point p = new Point(col, row);
                    CellWidget button =
    factory.createCellWidget(field.cell(p));
                    button.addActionListener(new FieldClickListener());
                    add(button);
                    cellWidgetMap.put(button, p);
                }
```

```
        }
        validate();
    }

    private Point getPointByButton(CellWidget cellWidget){
        return (Point) cellWidgetMap.get(cellWidget).clone();
    }

    private Map<Point, CellWidget> reverseCells() {
        HashMap<Point, CellWidget> reverseMap = new HashMap<>();
        for (Map.Entry<CellWidget, Point> entry :
cellWidgetMap.entrySet()) {
            reverseMap.put(entry.getValue(), entry.getKey());
        }
        return reverseMap;
    }

    private void drawLettersOnField() {
        for (Map.Entry<CellWidget, Point> entry :
cellWidgetMap.entrySet()) {
            entry.getKey().repaintButton();
        }
    }

    private void setEnabledButtons(boolean flag) {
        for (CellWidget cell : cellWidgetMap.keySet()) {
            cell.setEnabled(flag);
        }
    }

    private void setEnabledCellsWithLetters() {
        for (Map.Entry<CellWidget, Point> entry :
cellWidgetMap.entrySet()) {
            entry.getKey().setEnabledIfHasLetter();
        }
    }

    private void getNewActiveButtons() {
        setEnabledButtons(false);
        activeCells.clear();
        for (Cell c : field.cellsAdjacentToLetters()) {
            CellWidget widget = reverseCells().get(c.position());
            activeCells.add(widget);
        }
    }

    private void setEnabledActiveButtons(boolean flag){
        for (CellWidget w : activeCells){
            w.setEnabled(flag);
        }
    }

    private void setEnabledCellsAdjacentToLetters(Cell cell) {
        Map<Point, CellWidget> map = reverseCells();
        for (Cell c : field.cellsAdjacentToLetters(cell)) {
            map.get(c.position()).setEnabled(true);
```

```
        }
    }

    private void setDefaultExceptChosenOne(Cell cell){
        for (Map.Entry<CellWidget, Point> entry :
cellWidgetMap.entrySet()) {
            if (!entry.getValue().equals(cell.position())) {
                entry.getKey().setColorDefault();
            }
            else entry.getKey().setColorPlaced();
        }
    }

    private void setDefault(){
        for (Map.Entry<CellWidget, Point> entry :
cellWidgetMap.entrySet()) {
            entry.getKey().setColorDefault();
        }
    }

    private class FieldClickListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {

            CellWidget button = (CellWidget) e.getSource();
            button.setEnabled(false);

            // Ставим на поле букву текущего игрока
            Point p = getPointByButton(button);
            for (Cell c : field.cells()) {
                if (c.position().equals(p)) {
                    if (c.letter() == null) {
                        button.setColorPlaced();
                        model.activePlayer().setLetterTo(p);
                    } else {
                        button.setColorChosen();
                        model.activePlayer().chooseLetter(p);
                    }
                }
            }
        }
    }

    private class PlayerObserver implements PlayerActionListener{

        @Override
        public void letterIsPlaced(PlayerActionEvent e) {
            drawLettersOnField();
            setEnabledCellsWithLetters();
            setDefaultExceptChosenOne(e.letter().cell());
        }

        @Override
        public void letterIsReceived(PlayerActionEvent e) {
            setEnabledActiveButtons(true);
        }
```

```
        @Override
        public void turnIsSkipped(PlayerActionEvent e) {
            setEnabledButtons(false);
            setDefault();
            drawLettersOnField();
            getNewActiveButtons();
        }

        @Override
        public void letterOnFieldIsChosen(PlayerActionEvent e) {
            setEnabledButtons(false);
            setEnabledCellsAdjacentToLetters(e.letter().cell());
        }

        @Override
        public void turnIsOver(PlayerActionEvent e) {
            setEnabledButtons(false);
            setDefault();
            getNewActiveButtons();
        }

        @Override
        public void cancel(PlayerActionEvent e) {
            setEnabledButtons(false);
            setDefault();
            drawLettersOnField();
        }
    }

    private class GameObserver implements GameListener{

        @Override
        public void gameFinished(GameEvent e) {
            setDefault();
            setEnabledButtons(false);
        }

        @Override
        public void currentLetterIsChosen(GameEvent e) {

        }

        @Override
        public void dictionaryHasNotContainsWord(GameEvent e) {

        }

        @Override
        public void wordHasBeenComposed(GameEvent e) {

        }
    }

    private class MenuObserver implements MenuListener {
        @Override
```

```
        public void newGameStarted() {
            rebuildField();
            setEnabledButtons(false);
            getNewActiveButtons();
        }
    }
}


public class GamePanel extends JFrame {
    private FieldWidget fieldWidget;

    private AlphabetWidget alphabetWidget;

    private ActionButtonWidget actionButtonWidget;

    private WordListWidget wordListWidget;

    private  PlayerScoreWidget playerScoreWidget;

    private JMenuBar menu;
    private final String fileItems[] = new String[]{"New", "Exit"};

    private final String fieldSizes[] = new String[]{"5x5", "6x6",
"7x7", "8x8"};

    private final String difficultyLevels[] = new String[]{"Легко",
"Поле", "Алфавит", "Поле и алфавит"};

    private String selectedSize = null;
    private String selectedDiff = null;

    private GameModel model = new GameModel();

    public GamePanel() {
        super();
        this.setTitle("Балда");

        //Представление должно реагировать на изменение состояния
модели
        model.addGameListener(new GameObserver());

        setLayout(new BorderLayout());

        // Меню
        createMenu();
        setJMenuBar(menu);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Создаем игровое поле
        this.fieldWidget = new FieldWidget(model.field(), model,
this);
        fieldWidget.createField();
        add(fieldWidget, BorderLayout.WEST);
```

```
        // Создаем панель, которую затем поместим на основную
панель
        JPanel centerPanel = new JPanel(new GridLayout(2, 1));

        // Создаем виджет, отображающий буквы алфавита
        this.alphabetWidget = new AlphabetWidget(new
ComplicatedAlphabet(), model, this);
        alphabetWidget.buildLetterPanel();

        centerPanel.add(alphabetWidget);

        // Создаем виджет, отображающий кнопки действий
        this.actionButtonWidget = new ActionButtonWidget(model,
this);
        actionButtonWidget.buildActionButtonsPanel();
        centerPanel.add(actionButtonWidget);

        add(centerPanel, BorderLayout.CENTER);

        // Создаем панель, которую затем поместим на основную
панель
        JPanel eastPanel = new JPanel();
        eastPanel.setLayout(new BoxLayout(eastPanel,
BoxLayout.Y_AXIS));

        eastPanel.setBorder(new EmptyBorder(0,10,0,10));

        // Создаем виджет, отображающий текущий счет игроков
        this.playerScoreWidget = new PlayerScoreWidget(model,
this);
        playerScoreWidget.buildPlayerScorePane();
        eastPanel.add(playerScoreWidget);

        // Создаем виджет, отображающий списки слов, составленные
игроками
        this.wordListWidget = new WordListWidget(model, this);
        wordListWidget.buildPlayerWordsBoard();
        eastPanel.add(wordListWidget);

        add(eastPanel, BorderLayout.EAST);

        pack();
        setResizable(false);
        setVisible(true);

    }

    // ----------------------------- Создаем меню -----------------
-----------------

    private void createMenu() {

        menu = new JMenuBar();
        JMenu fileMenu = new JMenu("File");

        for (int i = 0; i < fileItems.length; i++) {
```

```
            JMenuItem item = new JMenuItem(fileItems[i]);
            item.setActionCommand(fileItems[i].toLowerCase());
            item.addActionListener(new NewMenuListener());
            fileMenu.add(item);
        }
        fileMenu.insertSeparator(1);

        menu.add(fileMenu);
    }

    public class NewMenuListener implements ActionListener {
        @Override
        public void actionPerformed(ActionEvent e) {
            String command = e.getActionCommand();
            if ("exit".equals(command)) {
                System.exit(0);
            }
            if ("new".equals(command)) {
                if (applySettings()) {
                    model.startGame();
                    fireNewGameStarted();
                }
            }
        }
    }

    private List<MenuListener> listeners = new ArrayList<>();

    public void addMenuListener(MenuListener l){ listeners.add(l);
}

    public void removeMenuListener(MenuListener l){
listeners.remove(l); }

    private void fireNewGameStarted(){
        for (MenuListener l : listeners){
            l.newGameStarted();
        }
    }

    private boolean applySettings() {
        HashMap<String, Integer> size = new HashMap<>();
        int index = 5;
        for (String str : fieldSizes) {
            size.put(str, index);
            index++;
        }

        JDialog settingsDialog = new JDialog();
        settingsDialog.setLayout(new GridLayout(3, 2));

settingsDialog.setModalityType(Dialog.ModalityType.APPLICATION_MODA
L);

        JLabel label1 = new JLabel("Размер поля: ");
```

```
        JLabel label2 = new JLabel("Сложность: ");

        JComboBox<String> comboBoxSize = new
JComboBox<>(fieldSizes);
        JComboBox<String> comboBoxDiff = new
JComboBox<>(difficultyLevels);

        JButton button1 = new JButton("Ок");
        JButton button2 = new JButton("Отмена");

        button1.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                selectedSize = (String)
comboBoxSize.getSelectedItem();
                selectedDiff = (String)
comboBoxDiff.getSelectedItem();

                System.out.println(selectedSize);
                System.out.println(selectedDiff);

                settingsDialog.dispose();
            }
        });

        button2.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                settingsDialog.dispose();
            }
        });

        settingsDialog.add(label1);
        settingsDialog.add(comboBoxSize);
        settingsDialog.add(label2);
        settingsDialog.add(comboBoxDiff);
        settingsDialog.add(button1);
        settingsDialog.add(button2);

        settingsDialog.pack();
        settingsDialog.setResizable(false);
        settingsDialog.setVisible(true);

        // After closing the dialog, program execution continues
here
        if (selectedSize != null && selectedDiff != null) {
            int heightAndWidth = size.get(selectedSize);
            model.field().setSize(heightAndWidth, heightAndWidth);
            switch (selectedDiff){
                case "Легко":
                    alphabetWidget.setAlphabet(new Alphabet());
                    model.field().setDiffLevel(1);
                    break;
                case "Поле":
                    alphabetWidget.setAlphabet(new Alphabet());
                    model.field().setDiffLevel(2);
```

```
                    break;
                case "Алфавит":
                    alphabetWidget.setAlphabet(new
ComplicatedAlphabet());
                    model.field().setDiffLevel(1);
                    break;
                case "Поле и алфавит":
                    alphabetWidget.setAlphabet(new
ComplicatedAlphabet());
                    model.field().setDiffLevel(2);
            }
            selectedSize = null;
            selectedDiff = null;

            return true;
        }
        return false;
    }

    // ------------------------- Реагируем на изменения модели ----
------------------
        private class GameObserver implements GameListener {

            @Override
            public void gameFinished(GameEvent e) {
                if (e.player() != null) {
                    String str = "Победил игрок" +
e.player().name();
                    JOptionPane.showMessageDialog(null, str,
"Победа!", JOptionPane.INFORMATION_MESSAGE);
                } else {
                    String str = "Ничья";
                    JOptionPane.showMessageDialog(null, str,
"Ничья", JOptionPane.INFORMATION_MESSAGE);
                }
            }
            @Override
            public void currentLetterIsChosen(GameEvent e) {}

            @Override
            public void dictionaryHasNotContainsWord(GameEvent e) {
                int result = JOptionPane.showConfirmDialog(null,
                        "Введённого слова нет в словаре.
Добавить?", "Добавление слова",
                        JOptionPane.YES_NO_OPTION);
                switch (result) {
                    case 0:

Dictionary.addWord(e.player().currentWord());
                        e.player().endTurn();
                        break;
                    case 1, -1:
                        e.player().cancel();
                        break;
                }
            }
```

```
            @Override
            public void wordHasBeenComposed(GameEvent e) {
                JOptionPane.showMessageDialog(null, "Введённое вам
слово уже было составлено", "Отмена",
JOptionPane.INFORMATION_MESSAGE);
                e.player().cancel();
            }
        }
    }



public class PlayerScoreWidget extends JPanel {
    private int firstPlayerScore = 0;

    private int secondPlayerScore = 0;

    private JLabel firstPlayerLabel = new JLabel();

    private JLabel secondPlayerLabel = new JLabel();

    public PlayerScoreWidget(GameModel model, GamePanel panel){
        model.addGameListener(new GameObserver());
        model.addPlayerActionListener(new PlayerObserver());
        panel.addMenuListener(new MenuObserver());
    }

    public void buildPlayerScorePane(){
        setLayout(new GridLayout(1, 2));
        setBorder(BorderFactory.createEmptyBorder(5,0,0,0));

        Dimension dimension = new Dimension(300, 50);

        setPreferredSize(dimension);
        setMinimumSize(dimension);
        setMaximumSize(dimension);

        firstPlayerLabel.setText("Счет первого игрока: " +
firstPlayerScore);
        secondPlayerLabel.setText("Счёт второго игрока: " +
secondPlayerScore);

        add(firstPlayerLabel);
        add(secondPlayerLabel);

        setVisible(true);
    }

    private class PlayerObserver implements PlayerActionListener {
        @Override
        public void letterIsPlaced(PlayerActionEvent e) {}
        @Override
        public void letterIsReceived(PlayerActionEvent e) {}
        @Override
        public void turnIsSkipped(PlayerActionEvent e) {}
```

```
        @Override
        public void letterOnFieldIsChosen(PlayerActionEvent e) {}
        @Override
        public void turnIsOver(PlayerActionEvent e) {
            if (e.currentWord() != null &&
!e.currentWord().isEmpty()) {
                if (e.player().name().equals("1")) {
                    firstPlayerScore += e.currentWord().length();
                    firstPlayerLabel.setText("Счет первого игрока:
" + firstPlayerScore);
                }
                else if (e.player().name().equals("2")) {
                    secondPlayerScore += e.currentWord().length();
                    secondPlayerLabel.setText("Счёт второго игрока:
" + secondPlayerScore);
                }
            }
        }
        @Override
        public void cancel(PlayerActionEvent e) {}
    }

    private class GameObserver implements GameListener {
        @Override
        public void gameFinished(GameEvent e) {
            firstPlayerScore = 0;
            secondPlayerScore = 0;
        }
        @Override
        public void currentLetterIsChosen(GameEvent e) {}
        @Override
        public void dictionaryHasNotContainsWord(GameEvent e) {}
        @Override
        public void wordHasBeenComposed(GameEvent e) {}
    }

    private class MenuObserver implements MenuListener {
        @Override
        public void newGameStarted() {
            firstPlayerScore = 0;
            secondPlayerScore = 0;
            firstPlayerLabel.setText("Счет первого игрока: " +
firstPlayerScore);
            secondPlayerLabel.setText("Счёт второго игрока: " +
secondPlayerScore);
        }
    }
}



public class WidgetFactory {
    private final Map<Cell, CellWidget> cells = new HashMap<>();


    //Создать виджет клетки
```

```
    public CellWidget createCellWidget(Cell cell){
        if(cells.containsKey(cell)) return cells.get(cell);

        CellWidget cellWidget = new CellWidget(cell);
        cells.put(cell, cellWidget);
        return cellWidget;
    }
}


public class WordListWidget extends JPanel {
    private DefaultListModel<String> firstPlayerList;

    private DefaultListModel<String> secondPlayerList;

    private JList<String> firstPlayerJList;

    private JList<String> secondPlayerJList;

    public WordListWidget(GameModel model, GamePanel panel){
        model.addPlayerActionListener(new PlayerObserver());
        model.addGameListener(new GameObserver());
        panel.addMenuListener(new MenuObserver());
    }

    public void buildPlayerWordsBoard() {
        setLayout(new GridLayout(1, 2));
        setBorder(BorderFactory.createEmptyBorder(0,0,5,0));

        Dimension dimension = new Dimension(300, 250);

        setPreferredSize(dimension);
        setMinimumSize(dimension);
        setMaximumSize(dimension);

        firstPlayerList = new DefaultListModel<>();
        secondPlayerList = new DefaultListModel<>();
        firstPlayerJList = new JList<>(firstPlayerList);
        secondPlayerJList = new JList<>(secondPlayerList);

        JScrollPane scrollPane1 = new
JScrollPane(firstPlayerJList);
        JScrollPane scrollPane2 = new
JScrollPane(secondPlayerJList);

        add(scrollPane1);
        add(scrollPane2);

        setVisible(true);
    }


    private class PlayerObserver implements PlayerActionListener{
        @Override
        public void letterIsPlaced(PlayerActionEvent e) {}
        @Override
```

```
        public void letterIsReceived(PlayerActionEvent e) {}
        @Override
        public void turnIsSkipped(PlayerActionEvent e) {}
        @Override
        public void letterOnFieldIsChosen(PlayerActionEvent e) {}
        @Override
        public void turnIsOver(PlayerActionEvent e) {
            if (e.currentWord() != null &&
!e.currentWord().isEmpty()) {
                if (e.player().name().equals("1"))
                    firstPlayerList.addElement(e.currentWord());
                else if (e.player().name().equals("2"))
                    secondPlayerList.addElement(e.currentWord());
            }
        }
        @Override
        public void cancel(PlayerActionEvent e) {}
    }

    private class GameObserver implements GameListener{
        @Override
        public void gameFinished(GameEvent e) {
            firstPlayerList.clear();
            secondPlayerList.clear();
        }
        @Override
        public void currentLetterIsChosen(GameEvent e) {}
        @Override
        public void dictionaryHasNotContainsWord(GameEvent e) {}
        @Override
        public void wordHasBeenComposed(GameEvent e) {}
    }

    private class MenuObserver implements MenuListener {
        @Override
        public void newGameStarted() {
            firstPlayerList.clear();
            secondPlayerList.clear();
        }
    }
}


public class Scrabble {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(GamePanel::new);
    }
}
```

5 Список использованной литературы и других источников

1. Простое объяснение принципов SOLID [Электронный ресурс] : Режим доступа: https://habr.com/ru/companies/vk/articles/412699/
2. Структуры данных в картинках. HashMap [Электронный ресурс] : Режим доступа: https://habr.com/ru/articles/128017/
3. Библиотека Swing [Электронный ресурс] : Режим доступа: https://java-online.ru/libs-swing.xhtml