|  |  |
| --- | --- |
| Описание: тов_знак_прав | МИНОБРНАУКИ РОССИИ  Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования  **«Санкт-Петербургский государственный электротехнический университет “ЛЭТИ” им. В.И. Ульянова (Ленина)» (СПбГЭТУ)** |
|  |  |

¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯¯

**­­­ОТЧЁТ**

**по проекту, выполняемому в рамках учебной практики**

**Выполнил:** Почанин Роман  
**Факультет:** КТИ  
**Группа:** 4382

Санкт-Петербург

2016

1. Формулировка задания:

Требуется написать программу, которая, по заданному положению юнитов(пешек и фигур) на шахматной доске, определяет, какие из клеток «бьются» и каким количеством юнитов.

1. Спецификация:

*Входные данные:*

На вход подаётся расположение шахматных юнитов (юнит - это фигура или пешка) на доске, задаваемое из файла, в следующем формате:

[именование фигуры] [положение]

Фигуры именуются следующим образом:

|  |  |
| --- | --- |
| Фигура | Обозначение |
| Король | K |
| Ферзь | Q |
| Ладья | R |
| Конь | N |
| Слон | B |
| Пешка | p |

Цвет задаётся префиксом - буквой w (белый) или b (черный). Положение задаётся стандартным для шахмат образом.

*Выходные данные:*

Рисунок доски с рассчитанными числовыми характеристиками клеток (см. рис. 1 и 2). Числовая характеристика каждой клетки определяется как разность A и B, где A - количество юнитов одной (белой) армии, атакающих клетку; B - количество юнитов другой (синей) армии, атакающих клетку. Король, в отличие от других юнитов, атакует клетку, на которой стоит. Нули для клеток, которые не атакуются никакими юнитами, выводить на рисунке не надо. Отрицательность или положительность характеритики клетки на рисунке следует показывать цветом: Если число положительное - белым, отрицательное – синим.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ♜ | ♞1 | ♝1 | ♚2 | ♛1 | ♝1 | ♞1 | ♜ |
| ♟1 | ♟1 | ♟1 | ♟4 | ♟4 | ♟1 | ♟1 | ♟1 |
| 2 | 2 | 3 | 2 | 2 | 3 | 2 | 2 |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 2 | 2 | 3 | 2 | 2 | 3 | 2 | 2 |
| ♙1 | ♙1 | ♙1 | ♙4 | ♙4 | ♙1 | ♙1 | ♙1 |
| ♖ | ♘1 | ♗1 | ♔2 | ♕1 | ♗1 | ♘1 | ♖ |

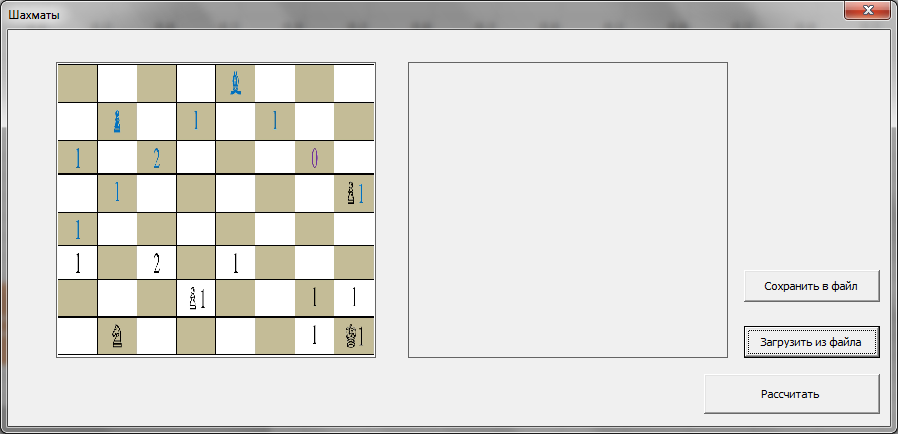
Рис. 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | ♝ |  |  |  |
|  | ♟ |  | 1 |  | 1 |  |  |
| 1 |  | 2 |  |  |  | 0 |  |
|  | 1 |  |  |  |  |  | ♙1 |
| 1 |  |  |  |  |  |  |  |
| 1 |  | 2 |  | 1 |  |  |  |
|  |  |  | ♙1 |  |  | 1 | 1 |
|  | ♘ |  |  |  |  | 1 | ♔1 |

Рис. 2

*Описание интерфейса:*

Интерфейс будет выглядеть следующим образом:



Таблица

Будет иметься возможность загрузить расстановку из файла, а также изменить её в специальном поле (таблице справа) и сохранить в файл.

1. План разработки

29.06.16 – показ первой версии (без возможности генерации таблицы по файлу, без возможности загрузки таблицы из файла, изменения размеров таблиц);

01.07.16 – показ второй версии (с интерфейсом, полностью соответствующим спецификации);

04.07.16 – показ финальной версии, полностью соответствующей спецификации и с необходимыми доработками.

1. Код

Основной класс, содержащий функцию main

**import** javax.swing.\*;  
**import** java.awt.\*;  
  
**public class** Main {  
 **public static void** main(String[] args) *//основной метод* {  
 MainWindow mainWindow = **new** MainWindow();  
  
 mainWindow.setTitle(**"Шахматы"**);  
 mainWindow.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);  
 mainWindow.setSize(1200, 520);  
 mainWindow.setMinimumSize(**new** Dimension(1000, 520));  
 mainWindow.setVisible(**true**);  
 }  
}

Класс, задающий внешний вид окна программы

**import** javax.swing.\*;  
**import** java.awt.\*;  
**import** java.awt.event.ActionEvent;  
**import** java.awt.event.ActionListener;  
**import** java.awt.event.ComponentEvent;  
**import** java.awt.event.ComponentListener;  
**import** java.io.\*;  
  
**import static** java.lang.Math.abs;  
**import static** java.lang.StrictMath.min;  
**import static** sun.swing.MenuItemLayoutHelper.*max*;  
  
*/\*\*  
 \* Created by Рома on 29.06.2016.  
 \*/***class** MainWindow **extends** JFrame {  
  
 */\*\*  
 \* Ниже описаны основные структуры данных, используемые в программе  
 \*/* **static** Cell [][] *ChessField* = **new** Cell[TableSize.***Hight***][TableSize.***Width***];  
 **static** String [][] *ChessField1* = **new** String[TableSize.***Hight***][TableSize.***Width***];  
 **static** String [][] *ChessField2* = **new** String[TableSize.***Hight***][TableSize.***Width***];  
 **static byte** [][] *Arr* = **new byte**[TableSize.***Hight***][TableSize.***Width***];  
  
 String[] **Column\_names** = {**"a"**, **"b"**, **"c"**, **"d"**, **"e"**, **"f"**, **"g"**, **"h"**};  
  
 JPanel **panel** = **new** JPanel();  
 JPanel **tables\_panel** = **new** JPanel();  
 JPanel **buttons\_panel** = **new** JPanel();  
  
 JPanel **table1\_panel** = **new** JPanel();  
 JPanel **table2\_panel** = **new** JPanel();  
  
 JPanel **button1\_panel** = **new** JPanel();  
 JPanel **button2\_panel** = **new** JPanel();  
 JPanel **button3\_panel** = **new** JPanel();  
  
 JTable **table2** = **new** JTable();  
  
 JLabel **label1** = **new** JLabel(**"Результат работы программы"**);  
 JLabel **label2** = **new** JLabel(**"Поле для входных данных"**);  
 JLabel **label3** = **new** JLabel(**"<html> Содержимое клетки задаётся в формате [цвет][фигура], цвет - одна буква: w - белая фигура, b - чёрная . <br> Шахматная фигура задаётся одной буквой: <br> K - король; Q - ферзь;<br> R - ладья; N - конь;<br> B - слон; p - пешка.</html>"**);  
  
 JButton **button1** = **new** JButton(**"Сохранить в файл"**);  
 JButton **button2** = **new** JButton(**"Загрузить из файла"**);  
 JButton **button3** = **new** JButton(**"Рассчитать"**);  
  
 JLayeredPane **chessPane**;  
 JPanel **newpanel** = **new** JPanel();  
 JLabel **chessPiece**;  
 JPanel **chessBoard**;  
 JPanel **squares**[][] = **new** JPanel[TableSize.***Hight*** + 1][TableSize.***Width*** + 1];  
  
 **class** Cell *// Класс, описывающий ячейку таблицы ChessField* {  
 **public** String **UnitName**;  
 **public int BeatenByWhite**, **BeatenByBlack**;  
  
 **public** Cell(String UnitName, **int** BeatenByWhite, **int** BeatenByBlack)  
 {  
 **this**.**UnitName** = UnitName;  
 **this**.**BeatenByWhite** = BeatenByWhite;  
 **this**.**BeatenByBlack** = BeatenByBlack;  
 }  
  
 **public** String getUnitName()  
 {  
 **return UnitName**;  
 }  
  
 **public int** getBeatenByWhite()  
 {  
 **return BeatenByWhite**;  
 }  
  
 **public int** getBeatenByBlack()  
 {  
 **return BeatenByBlack**;  
 }  
 }  
  
 **public abstract class** TableSize {  
 **public static final int *Width*** = 8;  
 **public static final int *Hight*** = 8;  
 **private** TableSize() {  
 }  
 }  
  
 */\*\*  
 \* Настройка врешнего вида окна  
 \*/* **public** MainWindow() {  
 **panel**.setLayout(**new** BorderLayout());  
 **panel**.add(**tables\_panel**, BorderLayout.***CENTER***);  
 **panel**.add(**buttons\_panel**, BorderLayout.***EAST***);  
  
 **tables\_panel**.setLayout(**new** GridLayout(1, 2));  
 **tables\_panel**.add(**table1\_panel**);  
 **tables\_panel**.add(**table2\_panel**);  
 tables\_config();  
  
 **table1\_panel**.setLayout(**new** BoxLayout(**table1\_panel**, BoxLayout.***Y\_AXIS***));  
  
 **label1**.setFont(**new** Font(**"Helvetica"**, Font.***BOLD***, 25));  
 **label1**.setPreferredSize(**new** Dimension(400,30));  
 **label1**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **table1\_panel**.add(**label1**);  
  
 **newpanel**.setLayout(**new** GridLayout(TableSize.***Hight***, TableSize.***Width***));  
 **newpanel**.setPreferredSize(**new** Dimension(400,400));  
 **chessPane** = **new** JLayeredPane();  
 getContentPane().add(**chessPane**);  
  
  
 **for** (**int** a = 0; a < TableSize.***Hight***; a++) *// Раскрашивание доски для результатов* {  
 **for** (**int** b = 0; b < TableSize.***Width***; b++)  
 {  
 **squares**[a][b] = **new** JPanel();  
  
 **if** ((a + b) % 2 == 0) {  
 **squares**[a][b].setBackground(**new** Color(238, 221, 187));  
 } **else** {  
 **squares**[a][b].setBackground(**new** Color(204, 136, 68));  
 }  
 **newpanel**.add(**squares**[a][b]);  
 }  
 }  
  
  
  
 **table1\_panel**.add(**newpanel**);  
  
 **label2**.setFont(**new** Font(**"Helvetica"**, Font.***BOLD***, 25));  
 **label2**.setPreferredSize(**new** Dimension(320,30));  
 **label3**.setPreferredSize(**new** Dimension(320,30));  
 **table2\_panel**.setLayout(**new** BoxLayout(**table2\_panel**, BoxLayout.***Y\_AXIS***));  
 **label2**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **table2**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **label3**.setAlignmentX(Component.***CENTER\_ALIGNMENT***);  
 **table2\_panel**.add(**label2**);  
 **table2\_panel**.add(**table2**);  
 **table2\_panel**.add(**label3**);  
  
 **buttons\_panel**.setLayout(**new** GridLayout(3, 1));  
 **buttons\_panel**.add(**button1\_panel**);  
 **buttons\_panel**.add(**button2\_panel**);  
 **buttons\_panel**.add(**button3\_panel**);  
  
 **button1\_panel**.setLayout(**new** BorderLayout());  
  
 **button2\_panel**.setLayout(**new** GridLayout(3, 1, 10, 10));  
 **button2\_panel**.add(**button1**, BorderLayout.***WEST***);  
 **button2\_panel**.add(**button2**, BorderLayout.***WEST***);  
 **button2\_panel**.add(**button3**, BorderLayout.***WEST***);  
  
 **button3\_panel**.setLayout(**new** BorderLayout());  
  
 **this**.getContentPane().add(**panel**);  
  
 **this**.**button1**.addActionListener(**new** MyButtonListener1());  
 **this**.**button2**.addActionListener(**new** MyButtonListener2());  
 **this**.**button3**.addActionListener(**new** MyButtonListener3());  
 }  
  
 */\*\*  
 \* Создание таблиц по массивам  
 \*/* **public void** tables\_config()  
 {  
 **table2** = **new** JTable(*ChessField2*, **Column\_names**);  
  
 **table2**.setFont(**new** Font(**"Helvetica"**, Font.***PLAIN***, 23));  
  
 **table2**.setRowHeight(40);  
  
 **table2**.setMinimumSize(**new** Dimension(320,320));  
 **table2**.setMaximumSize(**new** Dimension(320,320));  
  
 **for** (**int** i = 0; i < TableSize.***Width***; i++) {  
 **table2**.getColumnModel().getColumn(i).setCellRenderer(**new** Renderer\_table2());  
 }  
  
 **for** (**int** i = 0; i < TableSize.***Hight***; i++)  
 **for** (**int** j = 0; j < TableSize.***Width***; j++)  
 {  
 *ChessField1*[i][j] = **new** String(**""**);  
 *ChessField2*[i][j] = **new** String(**""**);  
 *Arr*[i][j] = 0;  
 *ChessField*[i][j] = **new** Cell(**""**,0, 0);  
 }  
 }  
  
 */\*\*  
 \* Ниже описаны функции обновления результатов для различных видов фигур  
 \*/* **public static void** KingUpdate(**boolean** isWhite, **int** row, **int** column) *// Обновление для короля* {  
 **for** (**int** i = row - 1; i <= row + 1; i++)  
 **for** (**int** j = column - 1; j <= column + 1; j++)  
 **if** (*areCorrectCoordinates*(i, j))  
 **if** (isWhite)  
 *ChessField*[i][j].**BeatenByWhite**++;  
 **else** *ChessField*[i][j].**BeatenByBlack**++;  
  
 }  
  
  
 **public static void** QueenUpdate(**boolean** isWhite, **int** row, **int** column) *// Обновление для ферзя* {  
 *RookUpdate*(isWhite, row, column);  
 *BishopUpdate*(isWhite, row, column);  
 }  
  
  
 **public static void** RookUpdate(**boolean** isWhite, **int** row, **int** column) *// Обновление для ладьи* {  
 **boolean** dir1 = **true**, dir2 = **true**, dir3 = **true**, dir4 = **true**;  
  
 **for** (**int** i = 1; (i < TableSize.***Width***) && dir1; i++) {  
 **if** (column + i < TableSize.***Width***) {  
 **if** (isWhite)  
 *ChessField*[row][column + i].**BeatenByWhite**++;  
 **else** *ChessField*[row][column + i].**BeatenByBlack**++;  
  
 **if** (*Arr*[row][column + i] == 1)  
 dir1 = **false**;  
 }  
 }  
  
 **for** (**int** i = 1; (i < TableSize.***Width***) && dir2; i++) {  
 **if** (column - i >= 0) {  
 **if** (isWhite)  
 *ChessField*[row][column - i].**BeatenByWhite**++;  
 **else** *ChessField*[row][column - i].**BeatenByBlack**++;  
  
 **if** (*Arr*[row][column - i] == 1)  
 dir2 = **false**;  
 }  
 }  
  
 **for** (**int** i = 1; (i < TableSize.***Hight***) && dir3; i++) {  
 **if** (row + i < TableSize.***Hight***) {  
 **if** (isWhite)  
 *ChessField*[row + i][column].**BeatenByWhite**++;  
 **else** *ChessField*[row + i][column].**BeatenByBlack**++;  
  
 **if** (*Arr*[row + i][column] == 1)  
 dir3 = **false**;  
 }  
 }  
  
 **for** (**int** i = 1; (i < TableSize.***Hight***) && dir4; i++) {  
 **if** (row - i >= 0) {  
 **if** (isWhite)  
 *ChessField*[row - i][column].**BeatenByWhite**++;  
 **else** *ChessField*[row - i][column].**BeatenByBlack**++;  
  
 **if** (*Arr*[row - i][column] == 1)  
 dir4 = **false**;  
 }  
 }  
 }  
  
  
 **public static void** KnightUpdate(**boolean** isWhite, **int** row, **int** column) *// Обновление для коня* {  
 **if** (*areCorrectCoordinates*(row + 1, column + 2))  
 **if** (isWhite)  
 *ChessField*[row + 1][column + 2].**BeatenByWhite**++;  
 **else** *ChessField*[row + 1][column + 2].**BeatenByBlack**++;  
  
 **if** (*areCorrectCoordinates*(row + 1, column - 2))  
 **if** (isWhite)  
 *ChessField*[row + 1][column - 2].**BeatenByWhite**++;  
 **else** *ChessField*[row + 1][column - 2].**BeatenByBlack**++;  
  
 **if** (*areCorrectCoordinates*(row - 1, column + 2))  
 **if** (isWhite)  
 *ChessField*[row - 1][column + 2].**BeatenByWhite**++;  
 **else** *ChessField*[row - 1][column + 2].**BeatenByBlack**++;  
  
 **if** (*areCorrectCoordinates*(row - 1, column - 2))  
 **if** (isWhite)  
 *ChessField*[row - 1][column - 2].**BeatenByWhite**++;  
 **else** *ChessField*[row - 1][column - 2].**BeatenByBlack**++;  
  
  
 **if** (*areCorrectCoordinates*(row + 2, column + 1))  
 **if** (isWhite)  
 *ChessField*[row + 2][column + 1].**BeatenByWhite**++;  
 **else** *ChessField*[row + 2][column + 1].**BeatenByBlack**++;  
  
 **if** (*areCorrectCoordinates*(row + 2, column - 1))  
 **if** (isWhite)  
 *ChessField*[row + 2][column - 1].**BeatenByWhite**++;  
 **else** *ChessField*[row + 2][column - 1].**BeatenByBlack**++;  
  
 **if** (*areCorrectCoordinates*(row - 2, column + 1))  
 **if** (isWhite)  
 *ChessField*[row - 2][column + 1].**BeatenByWhite**++;  
 **else** *ChessField*[row - 2][column + 1].**BeatenByBlack**++;  
  
 **if** (*areCorrectCoordinates*(row - 2, column - 1))  
 **if** (isWhite)  
 *ChessField*[row - 2][column - 1].**BeatenByWhite**++;  
 **else** *ChessField*[row - 2][column - 1].**BeatenByBlack**++;  
 }  
  
  
 **public static void** BishopUpdate(**boolean** isWhite, **int** row, **int** column) *// Обновление для слона* {  
 **boolean** dir1 = **true**, dir2 = **true**, dir3 = **true**, dir4 = **true**;  
  
 **for** (**int** i = 1; (i < *max*(TableSize.***Width***, TableSize.***Hight***)) && dir1; i++) {  
 **if** (row + i < TableSize.***Hight*** && column + i < TableSize.***Width***) {  
 **if** (isWhite)  
 *ChessField*[row + i][column + i].**BeatenByWhite**++;  
 **else** *ChessField*[row + i][column + i].**BeatenByBlack**++;  
  
 **if** (*Arr*[row + i][column + i] == 1)  
 dir1 = **false**;  
 }  
 }  
  
 **for** (**int** i = 1; (i < *max*(TableSize.***Width***, TableSize.***Hight***)) && dir2; i++) {  
 **if** (row - i >= 0 && column - i >= 0) {  
 **if** (isWhite)  
 *ChessField*[row - i][column - i].**BeatenByWhite**++;  
 **else** *ChessField*[row - i][column - i].**BeatenByBlack**++;  
  
 **if** (*Arr*[row - i][column - i] == 1)  
 dir2 = **false**;  
 }  
 }  
  
 **for** (**int** i = 1; (i < *max*(TableSize.***Width***, TableSize.***Hight***)) && dir3; i++) {  
 **if** (row + i < TableSize.***Hight*** && column - i >= 0) {  
 **if** (isWhite)  
 *ChessField*[row + i][column - i].**BeatenByWhite**++;  
 **else** *ChessField*[row + i][column - i].**BeatenByBlack**++;  
  
 **if** (*Arr*[row + i][column - i] == 1)  
 dir3 = **false**;  
 }  
 }  
  
 **for** (**int** i = 1; (i < *max*(TableSize.***Width***, TableSize.***Hight***)) && dir4; i++) {  
 **if** (row - i >= 0 && column + i < TableSize.***Width***)  
 {  
 **if** (isWhite)  
 *ChessField*[row - i][column + i].**BeatenByWhite**++;  
 **else** *ChessField*[row - i][column + i].**BeatenByBlack**++;  
  
 **if** (*Arr*[row - i][column + i] == 1)  
 dir4 = **false**;  
 }  
 }  
 }  
  
  
 **public static void** PawnUpdate(**boolean** isWhite, **int** row, **int** column) *// Обновление для пешки* {  
 **if** (isWhite)  
 {  
 **if** (row - 1 >= 0) {  
 **if** (column - 1 >= 0)  
 *ChessField*[row - 1][column - 1].**BeatenByWhite**++; *//сомнения!!!!!!!!!!!!!* **if** (column + 1 < TableSize.***Width***)  
 *ChessField*[row - 1][column + 1].**BeatenByWhite**++;  
 }  
 }  
 **else** {  
 **if** (row + 1 < TableSize.***Hight***) {  
 **if** (column - 1 >= 0)  
 *ChessField*[row + 1][column - 1].**BeatenByBlack**++;  
 **if** (column + 1 < TableSize.***Width***)  
 *ChessField*[row + 1][column + 1].**BeatenByBlack**++;  
 }  
 }  
 }  
  
  
 */\*\*  
 \* Ниже описана проверка корректности координат  
 \*/* **public static boolean** areCorrectCoordinates(**int** x, **int** y)  
 {  
 **if** (x >= 0 && x < TableSize.***Hight*** && y >= 0 && y < TableSize.***Width***)  
 **return true**;  
 **else  
 return false**;  
 }  
  
  
 */\*\*  
 \* Ниже описана обработка событий нажатий на кнопки  
 \*/* **private class** MyButtonListener1 **implements** ActionListener  
 {  
 @Override  
 **public void** actionPerformed(ActionEvent actionEvent)  
 {  
 *//C:\\Users\\Рома\\IdeaProjects\\Chess (v 2.0)\\out\\production\\Chess (v 2.0)\\table.txt* String fileName = **""**, text = **""**;  
 File file = **new** File(fileName);  
 JFileChooser fileopen = **new** JFileChooser();  
  
  
 **int** ret = fileopen.showDialog(**null**, **"Открыть файл"**);  
 **if** (ret == JFileChooser.***APPROVE\_OPTION***)  
 file = fileopen.getSelectedFile();  
  
  
 **try** {  
 **if**(!file.exists()){  
 file.createNewFile();  
 }  
 } **catch**(IOException e) {  
 **throw new** RuntimeException(e);}  
  
  
 **for** (**int** i = 0; i < TableSize.***Hight***; i++)  
 **for** (**int** j = 0; j < TableSize.***Width***; j++)  
 **if** (!*ChessField2*[i][j].equals(**""**))  
 {  
 text += *ChessField2*[i][j] + **" "** + i + **" "** + j + **"\n"**;  
 }  
  
 **try** {  
 PrintWriter out = **new** PrintWriter(file.getAbsoluteFile());  
 out.print(text);  
 out.close();  
 } **catch** (FileNotFoundException e) {  
 e.printStackTrace();}  
  
 }  
  
 }  
  
 **private class** MyButtonListener2 **implements** ActionListener  
 {  
 @Override  
 **public void** actionPerformed(ActionEvent actionEvent)  
 {  
 *//"C:\\Users\\Рома\\IdeaProjects\\Chess (v 2.0)\\out\\production\\Chess (v 2.0)\\table.txt* String fileName = **""**, textLine = **""**;  
 **int** x = 0, y = 0, c;  
 File file = **new** File(fileName);  
 JFileChooser fileopen = **new** JFileChooser();  
  
 **int** ret = fileopen.showDialog(**null**, **"Открыть файл"**);  
 **if** (ret == JFileChooser.***APPROVE\_OPTION***) {  
 file = fileopen.getSelectedFile();  
 fileName = file.getAbsolutePath();  
 }  
  
  
 **for** (**int** i = 0; i < TableSize.***Hight***; i++)  
 **for** (**int** j = 0; j < TableSize.***Width***; j++)  
 {  
 *ChessField1*[i][j] = **new** String(**""**);  
 *ChessField2*[i][j] = **new** String(**""**);  
 *Arr*[i][j] = 0;  
 *ChessField*[i][j] = **new** Cell(**""**,0, 0);  
 }  
  
 **try**(FileReader reader = **new** FileReader(fileName))  
 {  
 **char**[] buffer = **new char**[(**int**)file.length()];  
 reader.read(buffer);  
  
 **for** (**int** i = 0; i < (**int**)file.length() ; i++)  
 {  
 **if** (i % 7 == 0 || i % 7 == 1)  
 {  
 textLine += **""** + buffer[i];  
 } **else if** (i % 7 == 3)  
 {  
 x = buffer[i] - 48;  
 } **else if** (i % 7 == 5)  
 {  
 y = buffer[i] - 48;  
 *ChessField2*[x][y] = **new** String(textLine);  
 textLine = **""**;  
 }  
  
  
 }  
 }  
 **catch**(IOException ex){  
 System.***out***.println(ex.getMessage());  
 }  
  
 **table2**.repaint();  
 }  
  
 }  
  
 **private class** MyButtonListener3 **implements** ActionListener  
 {  
 @Override  
 **public void** actionPerformed(ActionEvent actionEvent)  
 {  
 **boolean** isWhite = **true**;  
  
 **for** (**int** i = 0; i < TableSize.***Hight***; i++)  
 **for** (**int** j = 0; j < TableSize.***Width***; j++)  
 {  
 **if** ( !*ChessField2*[i][j].equals(**""**))  
 *Arr*[i][j] = 1;  
 **else** {  
 *Arr*[i][j] = 0;  
 }  
  
 **squares**[i][j].removeAll();  
 **squares**[i][j].repaint();  
  
 *ChessField*[i][j].**BeatenByBlack** = 0;  
 *ChessField*[i][j].**BeatenByWhite** = 0;  
 }  
  
 **for** (**int** i = 0; i < TableSize.***Hight***; i++) *// Пересчёт результатов* **for** (**int** j = 0; j < TableSize.***Width***; j++)  
 **if** (*Arr*[i][j] == 1)  
 {  
 **if** (*ChessField2*[i][j].charAt(0) == **'w'**)  
 isWhite = **true**;  
 **else if** (*ChessField2*[i][j].charAt(0) == **'b'**)  
 isWhite = **false**;  
 **else** *errMessage*();  
  
 **switch** (*ChessField2*[i][j].charAt(1)) {  
 **case 'K'**:  
 *KingUpdate*(isWhite, i, j);  
 **break**;  
 **case 'Q'**:  
 *QueenUpdate*(isWhite, i, j);  
 **break**;  
 **case 'R'**:  
 *RookUpdate*(isWhite, i, j);  
 **break**;  
 **case 'N'**:  
 *KnightUpdate*(isWhite, i, j);  
 **break**;  
 **case 'B'**:  
 *BishopUpdate*(isWhite, i, j);  
 **break**;  
 **case 'p'**:  
 *PawnUpdate*(isWhite, i, j);  
 **break**;  
 **default**:  
 *errMessage*();  
 }  
 }  
  
  
 **for** (**int** i = 0; i < TableSize.***Hight***; i++) *// Вывод результатов в таблицу* **for** (**int** j = 0; j < TableSize.***Width***; j++)  
 {  
  
 **squares**[i][j].add(MakeLabel(i, j));  
 **squares**[i][j].validate();  
 }  
  
 }  
 }  
  
  
  
 */\*\*  
 \* Проверка существования файла  
 \*/* **private static void** exists(String fileName) **throws** FileNotFoundException {  
 File file = **new** File(fileName);  
 **if** (!file.exists()){  
 **throw new** FileNotFoundException(file.getName());  
 }  
 }  
  
  
 */\*\*  
 \* Вывод сообщения об ошибке  
 \*/* **private static void** errMessage()  
 {  
 System.***out***.print(**"Something wrong with the format!"**);  
 System.*exit*(1);  
 }  
  
  
 */\*\*  
 \* Создание Label-ов для вывода результатов  
 \*/* **public** JLabel MakeLabel(**int** i, **int** j)  
 {  
 Color color = **new** Color(0x73AADE);  
 JLabel answerLabel = **new** JLabel();  
 **int** result = MainWindow.*ChessField*[i][j].**BeatenByWhite** - MainWindow.*ChessField*[i][j].**BeatenByBlack**;  
  
 **if** (*Arr*[i][j] == 1)  
 {  
 ImageIcon icon = **new** ImageIcon(MainWindow.**class**.getResource(*ChessField2*[i][j] + **".png"**));  
 Image img = icon.getImage();  
 Image bi = img.getScaledInstance(**squares**[i][j].getWidth() - 10, **squares**[i][j].getHeight() - 10, Image.***SCALE\_SMOOTH***);  
 ImageIcon newIcon = **new** ImageIcon(bi);  
  
 answerLabel.setIcon(newIcon);  
  
 **if** (MainWindow.*ChessField*[i][j].**BeatenByWhite** > 0 || MainWindow.*ChessField*[i][j].**BeatenByBlack** > 0)  
 {  
 answerLabel.setLayout(**new** BorderLayout());  
 JLabel resultLabel = **new** JLabel(**""** + *abs*(result));  
 resultLabel.setFont(**new** Font(**"Helvetica"**, Font.***BOLD***, 25));  
 answerLabel.add(resultLabel, JLabel.***CENTER***);  
  
 **if** (result > 0) {  
 resultLabel.setForeground(Color.***WHITE***);  
 } **else if** (result < 0) {  
 resultLabel.setForeground(color);  
 } **else if** (result == 0) {  
 resultLabel.setForeground(Color.***LIGHT\_GRAY***);  
 }  
 }  
 }  
 **else** {  
 **if** (MainWindow.*ChessField*[i][j].**BeatenByWhite** > 0 || MainWindow.*ChessField*[i][j].**BeatenByBlack** > 0)  
 {  
 answerLabel.setLayout(**new** BorderLayout());  
 answerLabel.setFont(**new** Font(**"Helvetica"**, Font.***BOLD***, 25));  
 answerLabel.setText(**""** + *abs*(result));  
  
 **if** (result > 0) {  
 answerLabel.setForeground(Color.***WHITE***);  
 } **else if** (result < 0) {  
 answerLabel.setForeground(color);  
 } **else if** (result == 0) {  
 answerLabel.setForeground(Color.***LIGHT\_GRAY***);  
 }  
 }  
 }  
  
 **return** answerLabel;  
 }  
}

Пользовательский рендерер

**import** javax.swing.\*;  
**import** javax.swing.table.DefaultTableCellRenderer;  
**import** java.awt.\*;  
  
**public class** Renderer\_table2 **extends** DefaultTableCellRenderer  
{  
 @Override  
 **public** Component getTableCellRendererComponent(JTable table, Object value, **boolean** isSelected, **boolean** hasFocus, **int** row, **int** column)  
 {  
 Component cell = **super**.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);  
 Color color = **new** Color(204, 136, 68);  
 Color color2 = **new** Color(0xEEDDBB);  
  
 **if** ((row + column) % 2 == 0) *// раскраска доски* {  
 cell.setBackground(color2);  
 }  
 **else** {  
 cell.setBackground(color);  
 }  
  
 **return** cell;  
 }  
  
}

1. Тестирование









1. Заключение об успешности выполнения проекта:

В результате выполнения данного проекта, был получен опыт написания программ на языке объектно-ориентированного программирования Java. В ходе проекта была написана программа, целиком и полностью удовлетворяющая предъявляемым к ней требованиям, спецификации.