Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

Рязанский государственный радиотехнический университет

имени В.Ф. Уткина

Кафедра «САПР ВС»

К защите

«\_\_» \_\_\_\_\_\_\_\_ 2024 г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*подпись*

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К КУРСОВОЙ РАБОТЕ

На тему   
«Разработка графического приложения с использованием JavaFX»

по дисциплине

«Компьютерная графика»

**Выполнил**

ст. гр. 1413

Веркин Степан Александрович

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*дата сдачи на проверку, подпись*

**Проверил**

доцент

Митрошин Александр Александрович

\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*оценка дата защиты, подпись*

Рязань 2024

**Задание на курсовую работу**

Требуется разработать графическое приложение на языке Java с использованием библиотеки JavaFX.

Для разработки приложения была выбрана библиотека JavaFX, библиотека пользовательского интерфейса, предоставляющая набор инструментов и компонентов для разработки графических приложений на языке Java.

В целом, JavaFX предоставляет разработчикам широкий спектр инструментов для создания элегантных и современных пользовательских интерфейсов в своих приложениях Java.

Тема курсовой работы – графическое приложение для демонстрации возможностей выбранной библиотеки (JavaFX).

Приложение должно обладать следующими свойствами:

1. Приложение должно иметь графический интерфейс.
2. Приложение должно предоставлять пользователю возможность сыграть в игру «Тактические камень-ножницы-бумага» при использовании графического интерфейса

Содержание

[Введение 4](#_Toc155697039)

[1. Правила игры в «Тактические камень-ножницы-бумага» 5](#_Toc155697040)

[2. Окна 7](#_Toc155697041)

[3. Описание классов 10](#_Toc155697042)

[4. Вывод 11](#_Toc155697043)

[5. Список литературы и интернет ресурсов 12](#_Toc155697044)

[Приложение А. Код программы 13](#_Toc155697045)

[Класс Checkers 13](#_Toc155697046)

[Класс GameState 19](#_Toc155697047)

[Класс MainForms 19](#_Toc155697048)

[Класс MoveResult 23](#_Toc155697049)

[Класс MoveType 24](#_Toc155697050)

[Класс Piece 24](#_Toc155697051)

[Класс PieceTeam 27](#_Toc155697052)

[Класс PieceType 27](#_Toc155697053)

[Класс Tile 27](#_Toc155697054)

# Введение

В современном программировании Java выделяется как один из самых популярных языков программирования. Его мощные возможности и широкая поддержка делают его предпочтительным инструментом для различных приложений, включая область компьютерной графики. Java имеет богатый набор библиотек, специально разработанных для работы с графическими элементами. Библиотеки, такие как AWT (Abstract Window Toolkit) и JavaFX, позволяют разработчикам создавать эффектные и визуально привлекательные пользовательские интерфейсы, а также обеспечивают возможность работы с обработкой изображений, трехмерной графикой и анимацией.

В данной курсовой работе с помощью приложения с графическим пользовательским интерфейсом будут продемонстрированы возможности библиотеки JavaFX.

# Правила игры в «Тактические камень-ножницы-бумага»

По своей структуре «Тактические камень-ножницы-бумага» напоминают шахматы (наличием большого клеточного поля, уязвимого флажка и большего числа объектов) и классические «камень-ножницы-бумага» (системой превосходства определённых фигур над другими).

Цель игры – уничтожить флажок противника (поразить его своей фигурой в бою)

Каждый игрок располагает набором фигур: четыре камня, четыре бумаги, четверо ножниц, один камикадзе и один флажок.

Изначально игрок видит только расположение фигур соперника, но не их тип. Тип фигур раскрывается сопернику, только если фигура атаковала или была атакована.

За ход игрок может перемещать одну из своих фигур на соседнюю свободную клетку (вверх, влево, вправо или вниз) или атаковать фигуру соперника на соседней клетке, если есть такая возможность.

Одинаковые фигуры (за исключением камикадзе) при встрече в бою не делают ничего (если до боя встретившиеся в бою фигуры не были раскрыты, фигуры раскрываются).

При встрече в бою камня с ножницами фигура ножниц уничтожается, а фигура камня раскрывается.

При встрече в бою бумаги с камнем фигура камня уничтожается, а фигура камня раскрывается.

При встрече в бою ножниц с бумагой фигура бумаги уничтожается, а фигура ножниц раскрывается.

При встрече камикадзе с любым противником в бою обе фигуры уничтожаются.

При встрече флажка с любым противником в бою, кроме флажка, владелец флажка проигрывает.

Иллюстрированые виды противников указаны в таблице 1.

|  |  |
| --- | --- |
| Вид противника | Название |
|  | Флажок |
|  | Камикадзе |
|  | Бумага |
|  | Ножницы |
|  | Камень |
|  | Противник, тип которого ещё не исследован |

Таблица . Типы противников

# 2. Окна

В главном меню располагаются кнопки выхода из игры и начала новой игры (см. рисунок 1)

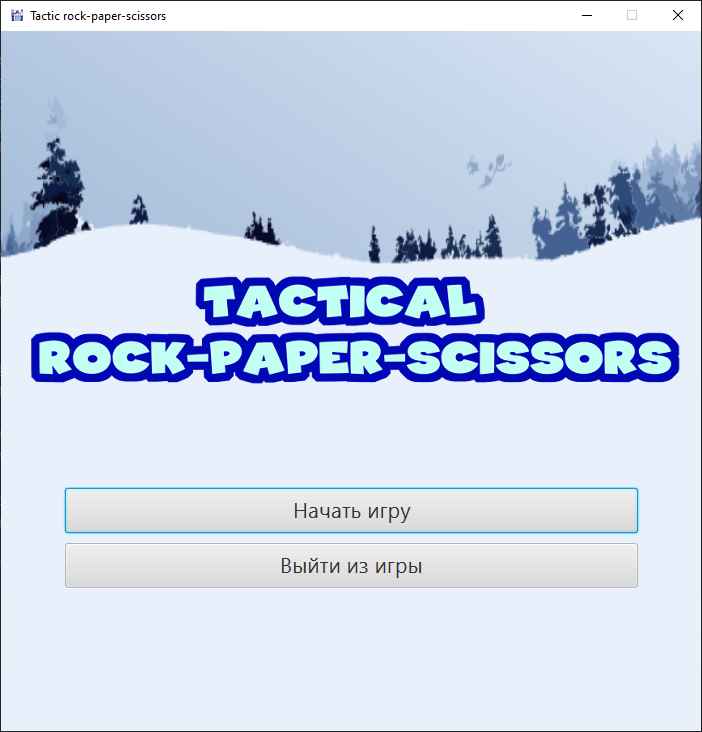


Рисунок . Окно главного меню

При запуске игры отображается игровое поле и вспомогательное меню с возможностью возврата в главное меню или перезапуска игры (см. рисунок 2)

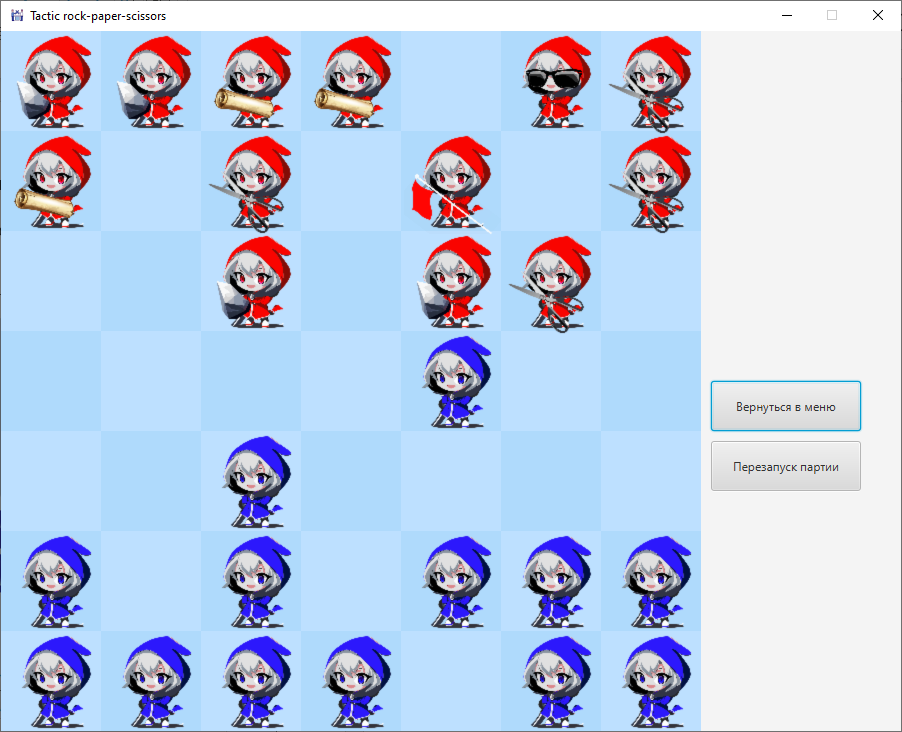


Рисунок . Окно игры

При завершении игры открывается окно с указанием победителя в текущем матче (см. рисунок 3)

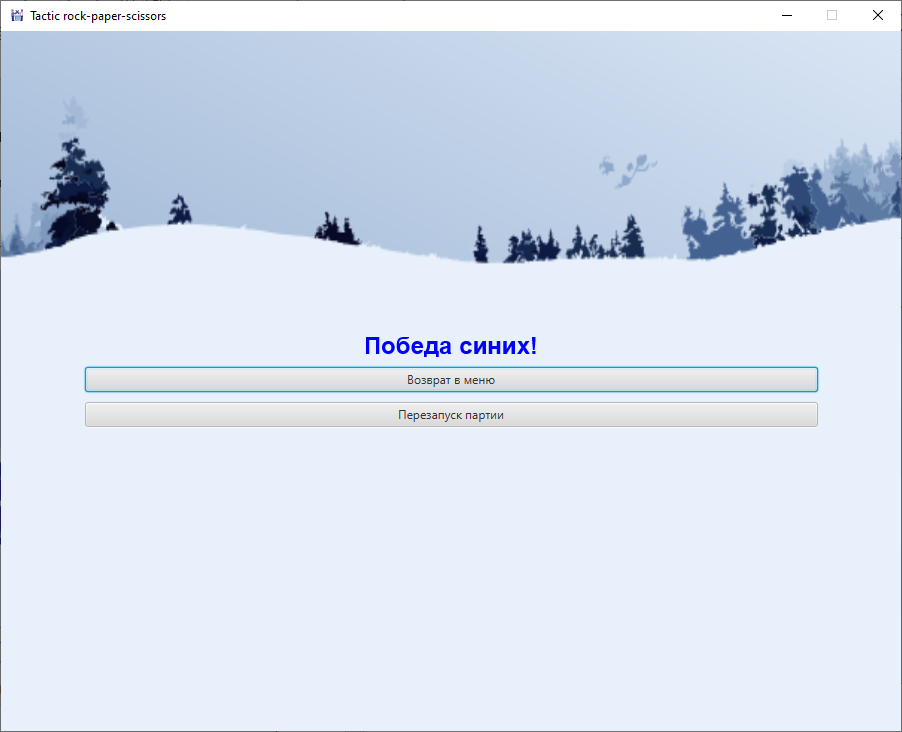


Рисунок . Окно завершения игры

# 3. Описание классов

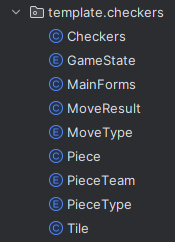


Рисунок . Структура проекта

Описание классов:

1. Checkers – класс, содержащий основную логику реализации игры.
2. GameState – класс-перечисление, содержащий константы, определяющие, находится игрок в меню или в игре.
3. MainForms – класс, содержащий разметку основных окон приложения и логику их взаимодействия.
4. MoveResult – класс, описывающий сущность перемещения фигур.
5. MoveType – класс-перечисление, содержащий виды перемещения фигур.
6. Piece – класс, описывающий сущность фигуры и содержащий методы её отрисовки.
7. PieceTeam – класс-перечисление, содержащий команды.
8. PieceType – класс-перечисление, содержащий типы фигур.
9. Tile – класс, описывающий сущность клетки игрового поля и содержащий методы их отрисовки.

# Вывод

При работе с данным курсовым проектом, мне удалось изучить возможности библиотеки JavaFX. JavaFX – полезная библиотека, с помощью которой можно создавать приложения с компьютерной графикой разной степени сложности, начиная от видеоигр и заканчивая профессиональными специализированными приложениями. Библиотека позволяет удобно работать как с внешними файлами по типу аудиофайлов или растровых изображений, так и своими возможностями может отображать элементы пользовательского интерфейса.

# Список литературы и интернет ресурсов

1.JavaFX для профессионалов / Хейкелл Д., Грейдс К., Смит Дж.: -Издательский дом "Питер", 2020. - 496 с.

2.Официальная документация JavaFX: сайт. – URL: https://openjfx.io/

3.JavaFX 11: Introduction by Example / Carl Dea, Gerrit Grunwald, José Pereda, Sean Phillips, Mark Heckler: -Apress, 2019. - 560 p.

4.Pro JavaFX 9: A Definitive Guide to Building Desktop, Mobile, and Embedded Java Clients / Johan Vos, Stephen Chin, Weiqi Gao, James Weaver, Dean Iverson: -Apress, 2017. - 579 p.

5.Mastering JavaFX 10: Build Advanced and visually stunning Java applications / Sergey Grinev: -Packt Publishing, 2018. - 446 p.

# Приложение А. Код программы

## Класс Checkers

package com.template.checkers;  
  
import javafx.scene.Group;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.layout.Pane;  
  
import javafx.scene.media.Media;  
import javafx.scene.media.MediaPlayer;  
import javafx.stage.Stage;  
  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.List;  
  
public class Checkers {  
  
 public static final int *TILE\_SIZE* = 100;  
 public static final int *WIDTH* = 7;  
 public static final int *HEIGHT* = 7;  
  
 public static final int *RIGHT\_MENU\_WIDTH* = *TILE\_SIZE* \* 2;  
 public static PieceTeam *currentPlayerTeam* = PieceTeam.*RED*;  
  
  
 private static Stage *stage* = null;  
  
  
 private static Tile[][] *board* = new Tile[*WIDTH*][*HEIGHT*];  
 private static Group *tileGroup* = new Group();  
 private static Group *pieceGroup* = new Group();  
  
  
 public static void startNewGame() {  
 *pieceGroup*.getChildren().clear();  
 *tileGroup*.getChildren().clear();  
 *board* = new Tile[*WIDTH*][*HEIGHT*];  
 }  
  
 private static void playSound(String soundName) {  
 String musicFile = "src/com.mycompany.rpsGame/com/sounds/" + soundName + ".mp3";  
 Media sound = new Media(new File(musicFile).toURI().toString());  
 MediaPlayer mediaPlayer = new MediaPlayer(sound);  
 mediaPlayer.play();  
 }  
  
 private static void reselectImagesForPieces(PieceTeam currentPlayerTeam) {  
 *pieceGroup*.getChildren().forEach(node -> {  
 Piece piece = (Piece) node;  
 try {  
 piece.reselectImage(currentPlayerTeam);  
 } catch (FileNotFoundException e) {  
 throw new RuntimeException(e);  
 }  
 });  
 }  
  
 public Checkers() {  
 }  
  
 private static List<PieceType> createShuffle() {  
 List<PieceType> pieces = new ArrayList<>();  
  
 for (int i = 0; i < 4; i++) {  
 pieces.add(PieceType.*STONE*);  
 }  
  
 for (int i = 0; i < 4; i++) {  
 pieces.add(PieceType.*SCISSORS*);  
 }  
  
 for (int i = 0; i < 4; i++) {  
 pieces.add(PieceType.*PAPER*);  
 }  
  
 pieces.add(PieceType.*KAMIKAZE*);  
 pieces.add(PieceType.*FLAG*);  
  
 Collections.*shuffle*(pieces);  
 return pieces;  
 }  
  
 public static void restartGame() throws FileNotFoundException {  
 *pieceGroup*.getChildren().clear();  
 *createPieces*();  
 *reselectImagesForPieces*(*currentPlayerTeam*);  
 }  
  
 private static void createTiles() {  
 for (int y = 0; y < *HEIGHT*; y++) {  
 for (int x = 0; x < *WIDTH*; x++) {  
 Tile tile = new Tile((x + y) % 2 == 0, x, y);  
 *board*[x][y] = tile;  
  
 *tileGroup*.getChildren().add(tile);  
 }  
 }  
  
 }  
  
 private static void createPieces() throws FileNotFoundException {  
  
 List<PieceType> reds = *createShuffle*();  
 List<PieceType> whites = *createShuffle*();  
  
 for (int y = 0; y < *HEIGHT*; y++) {  
 for (int x = 0; x < *WIDTH*; x++) {  
 Piece piece = null;  
  
 if (y <= 2) {  
 if (!reds.isEmpty()) {  
 piece = *makePiece*(reds.get(reds.size() - 1), PieceTeam.*RED*, x, y);  
 reds.remove(reds.size() - 1);  
 }  
 }  
  
 if (y >= 5) {  
 if (!whites.isEmpty()) {  
 piece = *makePiece*(whites.get(whites.size() - 1), PieceTeam.*WHITE*, x, y);  
 whites.remove(whites.size() - 1);  
 }  
 }  
  
 if (piece != null) {  
 *board*[x][y].setPiece(piece);  
 *pieceGroup*.getChildren().add(piece);  
 }  
 }  
 }  
 }  
  
 public static Parent createContent(Stage primaryStage) throws FileNotFoundException {  
  
 Checkers.*stage* = primaryStage;  
 Pane root = new Pane();  
 root.setPrefSize(*WIDTH* \* *TILE\_SIZE*, *HEIGHT* \* *TILE\_SIZE*);  
  
 root.getChildren().addAll(*tileGroup*, *pieceGroup*);  
  
 if (!MainForms.*gameStarted*) {  
 MainForms.*gameStarted* = true;  
  
 *createTiles*();  
  
 *createPieces*();  
 }  
  
 *reselectImagesForPieces*(*currentPlayerTeam*);  
  
  
 return root;  
 }  
  
  
 private static MoveResult tryMove(Piece piece, int newX, int newY) {  
  
 int x0 = *toBoard*(piece.getOldX());  
 int y0 = *toBoard*(piece.getOldY());  
 if ((newX > *WIDTH* - 1) || (newY > *WIDTH* - 1) || (newX < 0) || (newY < 0)) {  
 System.*out*.println("Move is impossible - too far away or beyond borders");  
 return new MoveResult(MoveType.*NONE*);  
 }  
  
 if (piece.getTeam() != *currentPlayerTeam*) {  
 return new MoveResult(MoveType.*NONE*);  
 }  
  
 if ((Math.*abs*(newX - x0) == 1 && newY == y0) || (Math.*abs*(newY - y0) == 1 && newX == x0)) {  
 if (*board*[newX][newY].hasPiece()) {  
 System.*out*.println("tile [" + newX + "; " + newY + "] has piece " + *board*[newX][newY].getPiece().getType());  
 if (*board*[newX][newY].getPiece().getTeam() != piece.getTeam()) {  
 System.*out*.println("BATTLE");  
 if (*board*[newX][newY].getPiece().getType() != piece.getType()) {  
 return new MoveResult(MoveType.*KILL*, *board*[x0][y0].getPiece(), *board*[newX][newY].getPiece());  
 } else {  
 return new MoveResult(MoveType.*EMPTY\_BATTLE*, *board*[x0][y0].getPiece(), *board*[newX][newY].getPiece());  
 }  
 } else {  
 return new MoveResult(MoveType.*NONE*);  
 }  
 } else {  
  
 return new MoveResult(MoveType.*NORMAL*);  
 }  
 }  
  
 return new MoveResult(MoveType.*NONE*);  
 }  
  
 private static int toBoard(double pixel) {  
 return (int) (pixel + *TILE\_SIZE* / 2) / *TILE\_SIZE*;  
 }  
  
  
 private static void killDefender(Tile fromTile, Tile toTile, Piece attacker, Piece defender) {  
 fromTile.setPiece(null);  
 toTile.setPiece(attacker);  
 *pieceGroup*.getChildren().remove(defender);  
 }  
  
 private static void killAttacker(Tile fromTile, Piece attacker) {  
 fromTile.setPiece(null);  
 *pieceGroup*.getChildren().remove(attacker);  
 }  
  
 private static Piece makePiece(PieceType type, PieceTeam team, int x, int y) throws FileNotFoundException {  
  
 Piece piece = new Piece(type, team, x, y);  
  
 piece.setOnMouseReleased(e -> {  
 int newX = *toBoard*(piece.getLayoutX());  
 int newY = *toBoard*(piece.getLayoutY());  
  
 MoveResult result = *tryMove*(piece, newX, newY);  
  
 int x0 = *toBoard*(piece.getOldX());  
 int y0 = *toBoard*(piece.getOldY());  
 Piece attacker = result.getPiece();  
 Piece defender = result.getDefenderPiece();  
  
 switch (result.getType()) {  
 case *NONE*:  
 *playSound*("error");  
 piece.abortMove();  
 break;  
 case *NORMAL*:  
 *changeCurrentPlayer*();  
 *playSound*("step");  
 piece.move(newX, newY);  
 *board*[x0][y0].setPiece(null);  
 *board*[newX][newY].setPiece(piece);  
 break;  
 case *EMPTY\_BATTLE*:  
  
  
 attacker.setExposed(true);  
 defender.setExposed(true);  
 *playSound*("draw");  
 piece.abortMove();  
 *changeCurrentPlayer*();  
 break;  
 case *KILL*:  
  
 piece.move(newX, newY);  
  
  
 PieceType attackerType = attacker.getType();  
 PieceType defenderType = defender.getType();  
  
 attacker.setExposed(true);  
 defender.setExposed(true);  
 *changeCurrentPlayer*();  
  
 Tile fromTile = *board*[x0][y0];  
 Tile toTile = *board*[newX][newY];  
  
 if (attackerType != PieceType.*FLAG* && defenderType == PieceType.*FLAG*) {  
  
 *killDefender*(fromTile, toTile, attacker, defender);  
 MainForms.*gameState* = GameState.*END*;  
 System.*out*.println("Владелец флажка проигрывает!");  
 PieceTeam loser = attacker.getTeam();  
 *finishGame*(loser);  
 break;  
 }  
  
  
 if (attackerType == PieceType.*KAMIKAZE* || defenderType == PieceType.*KAMIKAZE*) {  
 *playSound*("kamikaze");  
 *board*[x0][y0].setPiece(null);  
 *board*[newX][newY].setPiece(null);  
 *pieceGroup*.getChildren().remove(attacker);  
 *pieceGroup*.getChildren().remove(defender);  
 break;  
 }  
  
 if ((attackerType == PieceType.*STONE* && defenderType == PieceType.*SCISSORS*) ||  
 (attackerType == PieceType.*SCISSORS* && defenderType == PieceType.*PAPER*) ||  
 (attackerType == PieceType.*PAPER* && defenderType == PieceType.*STONE*)) {  
 *playSound*(attacker.getType().toString());  
 *killDefender*(fromTile, toTile, attacker, defender);  
 break;  
 }  
  
 if ((defenderType == PieceType.*STONE* && attackerType == PieceType.*SCISSORS*) ||  
 (defenderType == PieceType.*SCISSORS* && attackerType == PieceType.*PAPER*) ||  
 (defenderType == PieceType.*PAPER* && attackerType == PieceType.*STONE*)) {  
 *playSound*(defender.getType().toString());  
 *killAttacker*(fromTile, attacker);  
 break;  
 }  
  
 if (attackerType == PieceType.*FLAG* && defenderType != PieceType.*FLAG*) {  
 MainForms.*gameState* = GameState.*END*;  
  
 System.*out*.println("Владелец флажка проигрывает!");  
 PieceTeam loser = attacker.getTeam();  
 *killAttacker*(fromTile, attacker);  
 *finishGame*(loser);  
 break;  
 }  
  
 piece.abortMove();  
 *board*[x0][y0].setPiece(piece);  
 break;  
 }  
 });  
  
 return piece;  
 }  
  
 private static void finishGame(PieceTeam winner) {  
 *playSound*("endgame");  
 boolean redLost = (winner == PieceTeam.*WHITE*);  
 try {  
 Parent gameOverWindow = MainForms.*createGameOverWindow*(*stage*, redLost);  
 Scene scene = new Scene(gameOverWindow, *WIDTH* \* *TILE\_SIZE* + *RIGHT\_MENU\_WIDTH*, *HEIGHT* \* *TILE\_SIZE*);  
 *stage*.setScene(scene);  
 } catch (FileNotFoundException ex) {  
 throw new RuntimeException(ex);  
 }  
  
 }  
  
 private static void changeCurrentPlayer() {  
 if (currentPlayerTeam == PieceTeam.RED) currentPlayerTeam = PieceTeam.WHITE;  
 else currentPlayerTeam = PieceTeam.RED;  
 reselectImagesForPieces(currentPlayerTeam);  
  
  
 }  
  
  
}

## Класс GameState

package com.template.checkers;  
  
public enum GameState implements Comparable<GameState> {  
 *END*(-1), *PLAY*(1), *MENU*(0);  
  
  
 final int state;  
  
 GameState(int state) {  
 this.state = state;  
 }  
  
}

## Класс MainForms

package com.template.checkers;  
  
import javafx.application.Application;  
import javafx.geometry.Insets;  
import javafx.geometry.Pos;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.scene.control.Button;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.scene.layout.\*;  
import javafx.scene.paint.Color;  
import javafx.scene.text.Font;  
import javafx.scene.text.FontWeight;  
import javafx.scene.text.Text;  
import javafx.scene.text.TextBoundsType;  
import javafx.stage.Stage;  
  
  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
  
import static com.template.checkers.Checkers.\*;  
  
public class MainForms extends Application {  
  
  
 private static final int *BUTTON\_HEIGHT* = *TILE\_SIZE* / 2;  
 private static final int *BUTTON\_INTERVAL* = *TILE\_SIZE* / 4;  
 public static GameState *gameState* = GameState.*MENU*;  
  
 public static boolean *gameStarted* = false;  
  
 @Override  
 public void start(Stage primaryStage) throws FileNotFoundException {  
  
 Image icon = new Image(new FileInputStream("src/com.mycompany.rpsGame/com/images/icon.png"));  
 primaryStage.getIcons().add(icon);  
  
 primaryStage.setResizable(false);  
 primaryStage.setTitle("Tactic rock-paper-scissors");  
 Parent menu = *createMenu*(primaryStage);  
 Scene scene = new Scene(menu, *WIDTH* \* *TILE\_SIZE*, *HEIGHT* \* *TILE\_SIZE*);  
  
 primaryStage.setScene(scene);  
 primaryStage.show();  
 }  
  
 public static Parent createMenu(Stage primaryStage) throws FileNotFoundException {  
 StackPane root = new StackPane();  
  
 Image backgroundImage = new Image(new FileInputStream("src/com.mycompany.rpsGame/com/images/background.png"));  
 ImageView backgroundImageView = new ImageView(backgroundImage);  
 backgroundImageView.fitWidthProperty().bind(primaryStage.widthProperty());  
 backgroundImageView.fitHeightProperty().bind(primaryStage.heightProperty());  
 root.getChildren().add(backgroundImageView);  
  
 VBox menuContent = new VBox(10);  
 menuContent.setAlignment(Pos.*CENTER*);  
  
 System.*out*.println("Рисуем кнопки в меню");  
 Button startButton = new Button("Начать игру");  
 Button exitButton = new Button("Выйти из игры");  
  
 VBox buttonBox = new VBox(10);  
 buttonBox.setAlignment(Pos.*CENTER*);  
 buttonBox.setPadding(new Insets(20));  
  
 startButton.setStyle("-fx-font-size: 16pt;");  
 exitButton.setStyle("-fx-font-size: 16pt;");  
  
  
 Image logoImage = new Image(new FileInputStream("src/com.mycompany.rpsGame/com/images/mainlabel.png"));  
 ImageView logoImageView = new ImageView(logoImage);  
  
 double desiredWidth = *WIDTH* \* *TILE\_SIZE*;  
 double desiredHeight = *TILE\_SIZE* \* *HEIGHT*;  
  
  
 VBox logoContainer = new VBox(10);  
 logoContainer.setAlignment(Pos.*CENTER*);  
  
 logoImageView.setPreserveRatio(true);  
 logoImageView.setFitWidth(desiredWidth);  
 logoImageView.setFitHeight(desiredHeight);  
  
 logoContainer.getChildren().add(logoImageView);  
  
 startButton.prefWidthProperty().bind(primaryStage.widthProperty().multiply(0.8));  
 exitButton.prefWidthProperty().bind(primaryStage.widthProperty().multiply(0.8));  
  
 buttonBox.getChildren().addAll(logoContainer, startButton, exitButton);  
  
 startButton.setOnAction(event -> {  
 *gameState* = GameState.*PLAY*;  
 Scene gameScene;  
 try {  
 gameScene = new Scene(*createGameWindow*(primaryStage), *WIDTH* \* *TILE\_SIZE* + *RIGHT\_MENU\_WIDTH*, *HEIGHT* \* *TILE\_SIZE*);  
 } catch (FileNotFoundException e) {  
 throw new RuntimeException(e);  
 }  
 primaryStage.setScene(gameScene);  
 });  
  
 exitButton.setOnAction(event -> primaryStage.close());  
  
 menuContent.getChildren().addAll(buttonBox);  
 root.getChildren().add(menuContent);  
  
 buttonBox.setAlignment(Pos.*CENTER*);  
  
 return root;  
 }  
  
  
 private static Pane createGameWindow(Stage primaryStage) throws FileNotFoundException {  
 Pane gamePane = new Pane();  
 gamePane.setPrefSize(*WIDTH* \* *TILE\_SIZE* + *RIGHT\_MENU\_WIDTH*, *HEIGHT* \* *TILE\_SIZE*);  
  
 Pane gameField = (Pane) Checkers.*createContent*(primaryStage);  
 gameField.setLayoutX(0);  
 gameField.setLayoutY(0);  
  
 VBox menuButtons = new VBox();  
 menuButtons.setSpacing(10);  
 menuButtons.setAlignment(Pos.*CENTER*);  
  
 Button menuButton = *getMenuButton*(primaryStage);  
 Button restartButton = *getRestartButton*();  
  
 menuButtons.getChildren().addAll(menuButton, restartButton);  
  
 menuButtons.setLayoutX(*WIDTH* \* *TILE\_SIZE* + 10);  
 menuButtons.setLayoutY((gamePane.getPrefHeight() - menuButtons.getHeight()) / 2);  
  
  
  
 gamePane.getChildren().addAll(gameField, menuButtons);  
 return gamePane;  
 }  
  
 private static Button getRestartButton() {  
 Button restartButton = new Button("Перезапуск партии");  
 restartButton.setLayoutX(*WIDTH* \* *TILE\_SIZE* + *BUTTON\_INTERVAL*);  
 restartButton.setPrefHeight(*BUTTON\_HEIGHT*);  
 restartButton.setPrefWidth(*RIGHT\_MENU\_WIDTH* - *BUTTON\_INTERVAL* \* 2);  
 restartButton.setLayoutY(restartButton.getLayoutY() + *BUTTON\_HEIGHT* + *BUTTON\_INTERVAL*);  
 restartButton.setOnAction(e -> {  
 try {  
 Checkers.*restartGame*();  
 } catch (FileNotFoundException ex) {  
 throw new RuntimeException(ex);  
 }  
 });  
 return restartButton;  
 }  
  
 private static Button getMenuButton(Stage primaryStage) {  
 Button menuButton = new Button("Вернуться в меню");  
 menuButton.setPrefWidth(*RIGHT\_MENU\_WIDTH* - *BUTTON\_INTERVAL* \* 2);  
 menuButton.setLayoutX(*WIDTH* \* *TILE\_SIZE* + *BUTTON\_INTERVAL*);  
 menuButton.setPrefHeight(*BUTTON\_HEIGHT*);  
 menuButton.setLayoutY(10);  
 menuButton.setOnAction(e -> {  
 Scene scene;  
 try {  
 scene = new Scene(*createMenu*(primaryStage), *WIDTH* \* *TILE\_SIZE*, *HEIGHT* \* *TILE\_SIZE*);  
 primaryStage.setScene(scene);  
 } catch (FileNotFoundException ex) {  
 throw new RuntimeException(ex);  
 }  
  
 });  
 return menuButton;  
 }  
  
 static Parent createGameOverWindow(Stage primaryStage, boolean redLost) throws FileNotFoundException {  
 StackPane gameOverPane = new StackPane();  
  
 Image backgroundImage = new Image(new FileInputStream("src/com.mycompany.rpsGame/com/images/background.png"));  
 ImageView backgroundImageView = new ImageView(backgroundImage);  
 backgroundImageView.fitWidthProperty().bind(primaryStage.widthProperty());  
 backgroundImageView.fitHeightProperty().bind(primaryStage.heightProperty());  
 gameOverPane.getChildren().add(backgroundImageView);  
 gameOverPane.setPrefSize(*WIDTH* \* *TILE\_SIZE* + *RIGHT\_MENU\_WIDTH*, *HEIGHT* \* *TILE\_SIZE*);  
  
 String message = redLost ? "Победа синих!" : "Победа красных!";  
  
 Text gameOverText = new Text(message);  
 gameOverText.setFont(Font.*font*("Arial", FontWeight.*BOLD*, 24));  
 gameOverText.setFill(redLost ? Color.*BLUE* : Color.*RED*);  
 gameOverText.setBoundsType(TextBoundsType.*VISUAL*);  
  
 VBox buttonBox = new VBox(10);  
 buttonBox.setAlignment(Pos.*CENTER*);  
 buttonBox.setPadding(new Insets(20));  
  
 Button menuButton = new Button("Возврат в меню");  
 Button restartButton = *getRestartButton*(primaryStage);  
  
 menuButton.prefWidthProperty().bind(primaryStage.widthProperty().multiply(0.8));  
 restartButton.prefWidthProperty().bind(primaryStage.widthProperty().multiply(0.8));  
  
 buttonBox.getChildren().addAll(gameOverText, menuButton, restartButton);  
  
 menuButton.setOnAction(e -> {  
 Scene scene;  
 try {  
 scene = new Scene(*createMenu*(primaryStage), *WIDTH* \* *TILE\_SIZE*, *HEIGHT* \* *TILE\_SIZE*);  
 primaryStage.setScene(scene);  
 } catch (FileNotFoundException ex) {  
 throw new RuntimeException(ex);  
 }  
  
 });  
  
 gameOverPane.getChildren().add(buttonBox);  
 return gameOverPane;  
 }  
  
 private static Button getRestartButton(Stage primaryStage) {  
 Button restartButton = new Button("Перезапуск партии");  
 restartButton.setOnAction(e -> {  
 try {  
 Checkers.*restartGame*();  
 *gameState* = GameState.*PLAY*;  
 Scene gameScene;  
 gameScene = new Scene(*createGameWindow*(primaryStage), *WIDTH* \* *TILE\_SIZE* + *RIGHT\_MENU\_WIDTH*, *HEIGHT* \* *TILE\_SIZE*);  
 primaryStage.setScene(gameScene);  
 } catch (FileNotFoundException ex) {  
 throw new RuntimeException(ex);  
 }  
 });  
 return restartButton;  
 }  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }  
  
}

## Класс MoveResult

package com.template.checkers;  
  
public class MoveResult  
{  
 private MoveType type;  
  
 public MoveType getType() {  
 return type;  
 }  
  
 private Piece piece;  
 private Piece defenderPiece;  
  
 public Piece getPiece(){  
 return piece;  
 }  
  
 public Piece getDefenderPiece(){  
 return defenderPiece;  
 }  
  
 public MoveResult(MoveType type){  
 this(type, null);  
 }  
  
 public MoveResult(MoveType type, Piece piece) {  
 this.type = type;  
 this.piece = piece;  
 this.defenderPiece = null;  
 }  
  
 public MoveResult(MoveType type, Piece attackerPiece, Piece defenderPiece) {  
 this.type = type;  
 this.piece = attackerPiece;  
 this.defenderPiece = defenderPiece;  
 }  
}

## Класс MoveType

package com.template.checkers;  
  
public enum MoveType {  
 *NONE*, *NORMAL*, *KILL*, *EMPTY\_BATTLE*}

## Класс Piece

package com.template.checkers;  
  
import javafx.scene.effect.Blend;  
import javafx.scene.effect.BlendMode;  
import javafx.scene.effect.ColorInput;  
import javafx.scene.effect.GaussianBlur;  
import javafx.scene.image.Image;  
import javafx.scene.image.ImageView;  
import javafx.scene.layout.StackPane;  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Rectangle;  
  
import java.io.FileInputStream;  
import java.io.FileNotFoundException;  
  
import static com.template.checkers.Checkers.*TILE\_SIZE*;  
  
public class Piece extends StackPane {  
  
 private PieceType type;  
 private Integer STROKE\_WIDTH = 5;  
  
 private StackPane stackPane = new StackPane();  
  
 private Image image = new Image(new FileInputStream("src/com.mycompany.rpsGame/com/images/undefined.png"));  
  
 private PieceTeam team = PieceTeam.*UNDEFINED*;  
 private double mouseX, mouseY;  
 private double oldX, oldY;  
 private boolean visibility = false;  
  
 public boolean getVisibility() {  
 return visibility;  
 }  
  
 public PieceType getType() {  
 return type;  
 }  
  
 public PieceTeam getTeam() {  
 return team;  
 }  
  
 public double getOldX() {  
 return oldX;  
 }  
  
 public double getOldY() {  
 return oldY;  
 }  
  
 private void selectImage(PieceType type, PieceTeam team, PieceTeam currentPlayerColor) throws FileNotFoundException {  
 String commandPrefix = "";  
 String typeSuffix = "";  
  
 switch (team) {  
 case *RED*:  
 commandPrefix = "red";  
 break;  
 case *WHITE*:  
 commandPrefix = "blue";  
 break;  
 }  
  
 switch (type) {  
 case *STONE*:  
 typeSuffix = "Stone";  
 break;  
 case *SCISSORS*:  
 typeSuffix = "Scissors";  
 break;  
 case *PAPER*:  
 typeSuffix = "Paper";  
 break;  
 case *KAMIKAZE*:  
 typeSuffix = "Kamikaze";  
 break;  
 case *FLAG*:  
 typeSuffix = "Flag";  
 break;  
 }  
  
 if (!this.getVisibility() && (currentPlayerColor != this.team)) {  
 typeSuffix = "";  
 }  
  
 String imagePath = "src/com.mycompany.rpsGame/com/images/" + commandPrefix + typeSuffix + ".png";  
 this.image = new Image(new FileInputStream(imagePath));  
 }  
  
  
 public void reselectImage(PieceTeam currentPlayerTeam) throws FileNotFoundException {  
 this.selectImage(type, team, currentPlayerTeam);  
 ImageView imageView = (ImageView) stackPane.getChildren().get(0);  
 GaussianBlur gaussianBlur = new GaussianBlur(1.5);  
 imageView.setEffect(gaussianBlur);  
 imageView.setImage(this.image);  
 }  
  
 public Piece(PieceType type, PieceTeam team, int x, int y) throws FileNotFoundException {  
 this.type = type;  
 this.team = team;  
 this.visibility = false;  
 move(x, y);  
  
  
 this.stackPane = new StackPane();  
 this.selectImage(type, team, team);  
  
 ImageView imageView = new ImageView(this.image);  
 imageView.setSmooth(true);  
 imageView.setFitWidth(*TILE\_SIZE*);  
 imageView.setFitHeight(*TILE\_SIZE*);  
  
 Rectangle rectangle = new Rectangle(*TILE\_SIZE*, *TILE\_SIZE*);  
 rectangle.setFill(Color.*TRANSPARENT*);  
 rectangle.setStroke(Color.*TRANSPARENT*);  
 rectangle.setStrokeWidth(STROKE\_WIDTH);  
  
 stackPane.getChildren().addAll(imageView, rectangle);  
 getChildren().addAll(stackPane);  
  
 setOnMouseEntered(e -> rectangle.setStroke(Color.*BLUE*));  
  
 setOnMouseExited(e -> rectangle.setStroke(Color.*TRANSPARENT*));  
  
 setOnMousePressed(e -> {  
 rectangle.setStroke(Color.*RED*);  
 mouseX = e.getSceneX();  
 mouseY = e.getSceneY();  
 });  
  
 setOnMouseDragged(e -> relocate((e.getSceneX() - mouseX) + oldX, e.getSceneY() - mouseY + oldY));  
 }  
  
 public void move(int x, int y) {  
 oldX = x \* *TILE\_SIZE*;  
 oldY = y \* *TILE\_SIZE*;  
 relocate(oldX, oldY);  
 }  
  
 public void abortMove() {  
 relocate(oldX, oldY);  
 }  
  
 public boolean isExposed() {  
 return this.visibility;  
 }  
  
 public void setExposed(boolean visibility) {  
 this.visibility = visibility;  
 }  
}

## Класс PieceTeam

package com.template.checkers;  
  
public enum PieceTeam implements Comparable<PieceTeam> {  
 *RED*(1), *WHITE*(-1), *UNDEFINED*(0);  
  
  
 final int team;  
  
 PieceTeam(int team) {  
 this.team = team;  
 }  
  
}

## Класс PieceType

package com.template.checkers;  
  
public enum PieceType implements Comparable<PieceType> {  
 *RED*(1), *WHITE*(-1),  
  
 *STONE*(10), *SCISSORS*(11), *PAPER*(12), *KAMIKAZE*(13), *FLAG*(14);  
  
  
 final int moveDir;  
  
 PieceType(int moveDir) {  
 this.moveDir = moveDir;  
 }  
  
 @Override  
 public String toString() {  
 return switch (this.moveDir) {  
 case 10 -> "stone";  
 case 11 -> "scissors";  
 case 12 -> "paper";  
 case 13 -> "kamikaze";  
 default -> "error";  
 };  
 }  
  
  
}

## Класс Tile

package com.template.checkers;  
  
import javafx.scene.paint.Color;  
import javafx.scene.shape.Rectangle;  
  
public class Tile extends Rectangle {  
  
 private Piece piece;  
  
 public boolean hasPiece() {  
 return piece != null;  
 }  
  
 public Piece getPiece() {  
 return piece;  
 }  
  
 public void setPiece(Piece piece) {  
 this.piece = piece;  
 }  
 public Tile(boolean light, int x, int y) {  
 setWidth(Checkers.*TILE\_SIZE*);  
 setHeight(Checkers.*TILE\_SIZE*);  
  
 relocate(x \* Checkers.*TILE\_SIZE*, y \* Checkers.*TILE\_SIZE*);  
  
 setFill(light ? Color.*valueOf*("#bde0ff") : Color.*valueOf*("#afdafc"));  
 }  
}