МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и
автоматизированных систем

КУРСОВАЯ РАБОТА

по дисциплине: **Объектно-ориентированное программирование**
по теме: “Игра: Шахматы”

Автор работы_____Дмитриев Андрей Александрович ПВ-223

Руководитель проекта_____Черников Сергей Викторович

Оценка_____

Белгород 2024

Содержание

Введение

Курсовой проект посвящен разработке классической игры «шахматы» с интерфейсом и настройками игрового процесса на языке программирования C++ с библиотекой QT. Игра будет сопровождаться однооконным пользовательским интерфейсом, который будет предполагать место для горизонтального расширения.

Целью данной работы является создание шахматного движка с возможностью настраивать игру в некоторых режимах. Предполагается несколько режимов, реализовать один из них: игра друг против друга.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработка движка игры:
    a. Определить фигуры, рассчитываемые с помощью битбордов.
    b. Описать связь и взаимодействия битбордов и поля.
2. Реализация пользовательского интерфейса:
    a. Корректно отслеживать события мыши и выводить информацию о действиях фигур на доске.
    b. Создать «шахматные часы».
    c. Текстом выводить перемещения фигур по доске.
    d. Остальной интерфейс приложения.
3. Разработка интерфейса настроек:
    a. Определить формат сохранения.
    b. Создать пользовательский интерфейс для задания настроек.

1. Объектная декомпозиция

1.1 Диаграмма объектов

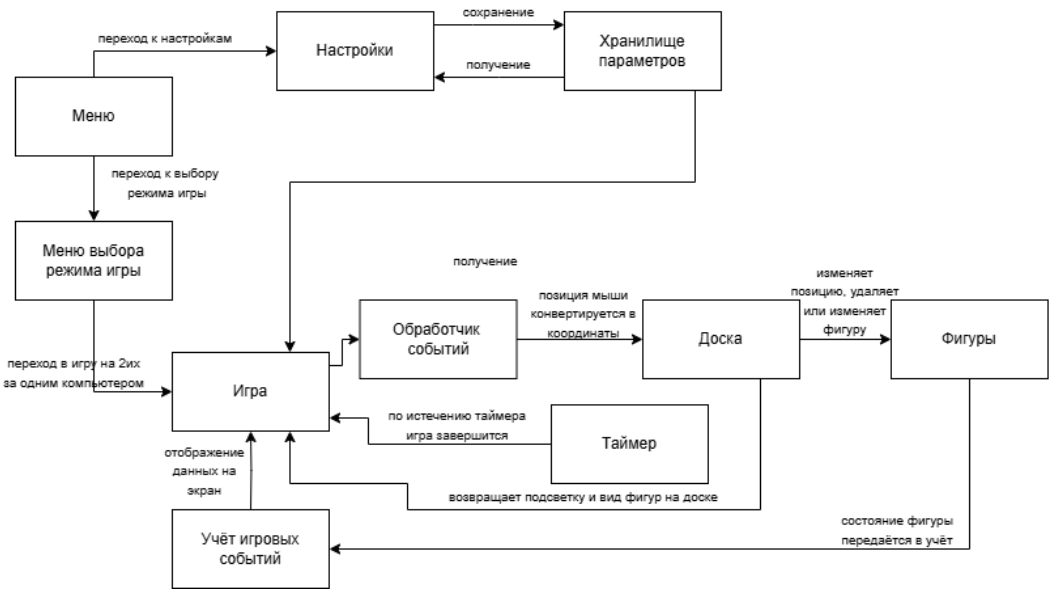

Рисунок 1. Диаграмма объектов

1.2 Диаграмма классов

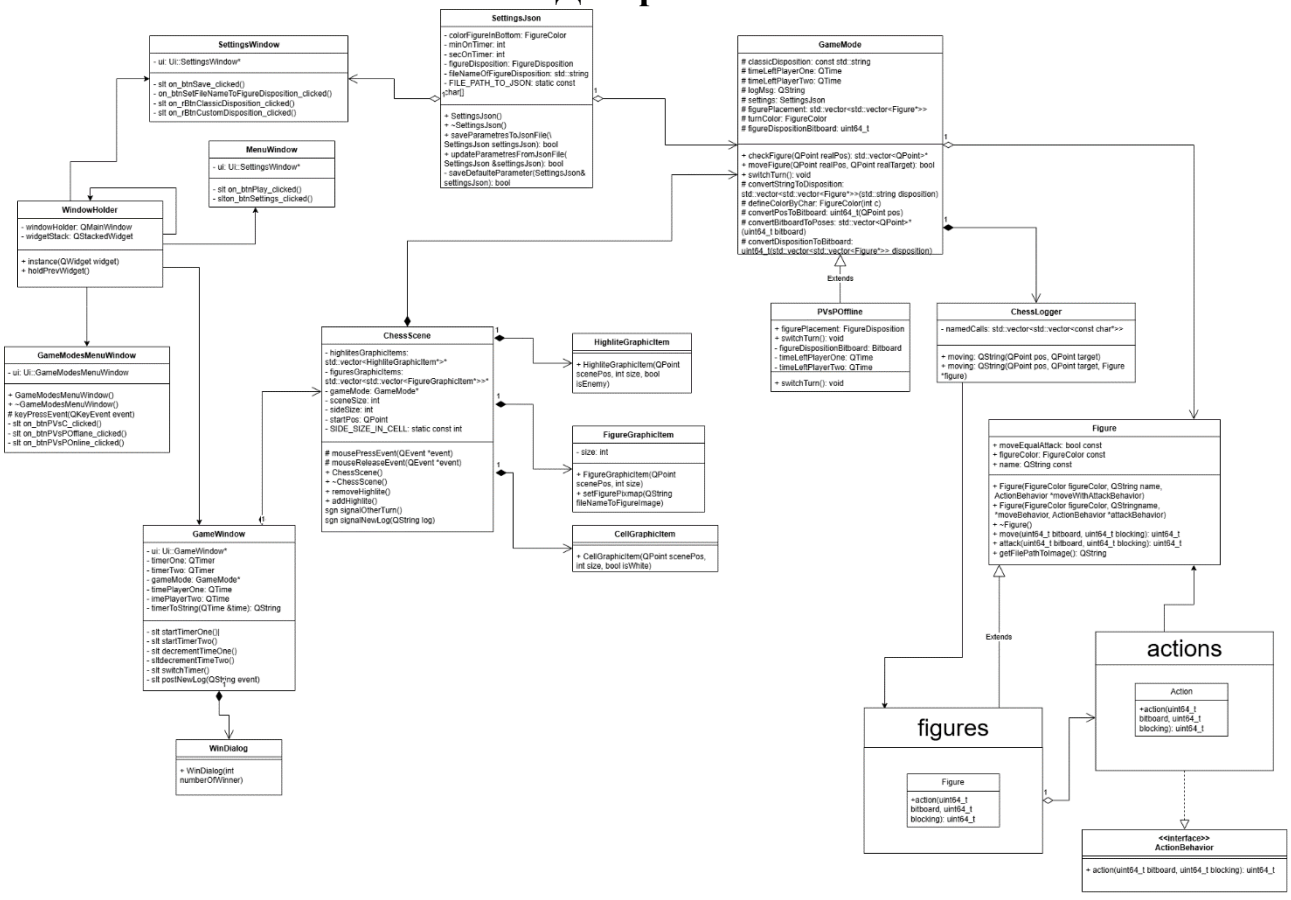

Рисунок 2. Диаграмма классов.

2. Разработка движка игры

2.1 Структура игры

Движок реализован с использованием концепции битбордов – представление шахмат в виде 64 битного беззнакого числа. В проекте эта концепция не будет использоваться в полной мере, то есть расчёт фигур и представление всей доски будет выполняться по битборам, а информация об отдельных фигурах на доске будет представляться матрицей. Проект реализован на языке программирования C++ с использованием принципов ООП. Движение фигур будет описано паттерном «Стратегия» и фигуры будут агрегировать их. Реализован режим игра против другого игрока.

2.2 Фигуры и описание движения

2.2.1 Класс ActionBehavior

Класс ActionBehavior представляет интерфейс для задания действий, а также он включает вспомогательные методы и константы.

В рассчёте ходов фигур используются битборды. Битборд – это массивная двоичная структура данных, которая используется для представления расположения фигур на шахматной доске. Каждая клетка доски представлена одним битом в битборде, а значение бита указывает, занята ли клетка фигурой. Операции с битбордами выполняются гораздо быстрее, чем операции с традиционными многомерными массивами.

```
class ActionBehavior {
public:
    virtual uint64_t action(uint64_t bitboard, uint64_t blocking) {
        return 0ULL;
    }

protected:
    static const uint64_t RANK_8 = 0xFF00000000000000;
    static const uint64_t RANK_7 = 0x00FF000000000000;
    static const uint64_t RANK_6 = 0x0000FF0000000000;
    static const uint64_t RANK_5 = 0x000000FF00000000;
    static const uint64_t RANK_4 = 0x00000000FF000000;
    static const uint64_t RANK_3 = 0x0000000000FF0000;
    static const uint64_t RANK_2 = 0x000000000000FF00;
    static const uint64_t RANK_1 = 0x00000000000000FF;
    static const uint64_t FILE_A = 0x0101010101010101;
```

```
    static const uint64_t FILE_B = 0x0202020202020202;
    static const uint64_t FILE_H = 0x8080808080808080;
    static const uint64_t FILE_G = 0x7070707070707070;

    static const uint64_t EMPTY_MASK = 0ULL;

    static uint64_t genRay(
        std::function<uint64_t(uint64_t)> bitShift,
        uint64_t file,
        uint64_t bitboard,
        uint64_t blocking) {
        uint64_t flood = bitboard;
        uint64_t empty = ~blocking & ~file;
        flood |= bitboard = (bitShift(bitboard)) & empty;
        flood |= bitboard = (bitShift(bitboard)) & empty;
        flood |= bitboard = (bitShift(bitboard)) & empty;
        flood |= bitboard = (bitShift(bitboard)) & empty;
        flood |= bitboard = (bitShift(bitboard)) & empty;
        flood |=            (bitShift(bitboard)) & empty;
        return              (bitShift(flood)) & ~file;
    }
};
```

Листинг 1. Класс ActionBehavior

Метод action() является реализуемым и принимает битборд с расположением фигуры и битборд с блокирующими фигурами и возвращает соответственно битборд с возможными ходами. Например для Лошади (Листинг 2) составление хода будут высчитываться таким образом. FILE_A – это заполнение по линии «A».

```
class HorseAction : public ActionBehavior {
public:
    uint64_t action(uint64_t bitboard, uint64_t blocking) override {
        const uint64_t notAFile = ~FILE_A;
        const uint64_t notABFile = notAFile & ~FILE_B;
        const uint64_t notHFile = ~FILE_H;
        const uint64_t notGHFile = notHFile & ~FILE_G;
        return (bitboard << 17) & notAFile | (bitboard << 10) & notABFile |
(bitboard >> 6) & notABFile | (bitboard >> 15) & notAFile |
               (bitboard << 15) & notHFile | (bitboard << 6) & notGHFile |
(bitboard >> 10) & notGHFile | (bitboard >> 17) & notHFile;
    }
};
```

Листинг 2. Класс HorseAction

2.2.2 Класс Figure

Класс Figure представляет базовый класс для всех типов фигур. Он в конструкторе требует задать экземпляр действия или действий на случай движения и атаки.

```
class Figure {
public:
    bool const moveEqualAttack;
    FigureColor const figureColor;
    QString const name;

    Figure(figureColor,
            QString name,
            ActionBehavior *moveWithAttackBehavior);

    Figure(figureColor,
            QString name,
            ActionBehavior *moveBehavior,
            ActionBehavior *attackBehavior
        );

    uint64_t move(uint64_t bitboard, uint64_t blocking);

    uint64_t attack(uint64_t bitboard, uint64_t blocking);

    ~Figure();

    QString getFilePathToImage();

protected:
    QString filePathToImage;

    void setPathToImage(FigureColor figureColor, const char* name);

private:
    ActionBehavior *movePerform;
    ActionBehavior *attackPerform;
};
```

Листинг 3. Класс Figure.

Пример конструирования той же фигуры Лошади в листинге 4. Пользователю потребуется всего лишь указать цвет фигуры.

```
class Horse : public Figure {
public:
    Horse(FigureColor figureColor)
        : Figure(figureColor, "horse", new HorseAction()) {
        setPathToImage(figureColor, "H");
    }
};
```

Листинг 4. Класс Horse.

2.3.1 Класс GameMode

Класс GameMode содержит информацию о доске и отвечает за их передвижение. Класс имеет множество методов для конвертирования игровых

досок из-за наличия различных представлений: битборда, матрица фигур, матрица графических объектов.

Метод checkFigure() запрашивает у фигуры, стоящей по передаваемой позиции, битборд и далее конвертирует его в координаты, удаляя фигуры своего цвета, чтобы на их место нельзя было встать в отличие от фигур противоположного цвета.

Метод moveFigure() перемещает фигуру удаляя фигуру по новой позиции, Метод не выполняет проверки на возможность хода, так как предполагается, что позиции будут проверены.

Абстрактный метод switchTurn передаёт ход другому игроку. Это действие неопределенно, так как подтверждение передачи хода может иметь разную логику, например, для локальной и онлайн игры.

```
class GameMode {
public:
    GameMode();
    bool isConteinFigureByPos(QPoint realPos);
    std::vector<std::vector<Figure*>> getFigurePlacementInv();
    virtual std::vector<QPoint> *checkFigure(QPoint realPos);
    virtual bool moveFigure(QPoint realPos, QPoint realTarget);
    QString getNewLogMsg();
    virtual void switchTurn();
    static int invCoord(int coord);
    QTime timeLeftPlayerOne;
    QTime timeLeftPlayerTwo;

protected:
    QString logMsg;
    SettingsJson settings;
    std::vector<std::vector<Figure*>> figurePlacement;
    FigureColor turnColor = FigureColor::WHITE;
    uint64_t figureDispositionBitboard;
    std::vector<QPoint> *getPosesWithoutAlly(QPoint& pos,
std::function<uint64_t(uint64_t, uint64_t)> action);
    static const int BOARD_SIDE = 8;
    static uint64_t convertPosToBitboard(QPoint pos);
    static std::vector<QPoint>* convertBitboardToPoses(uint64_t bitboard);
    static uint64_t convertDispositionToBitboard(std::vector<std::vector<Figure*>>
disposition);
};
```

Листинг 5. Класс GameMode.

3. Пользовательский интерфейс

3.1 Взаимодействие игрока с интерфейсом игры

3.1.1 Класс ChessScene

Класс ChessScene наследуется от класса, представляющего инструменты для отрисовки различных графических объектов. В конструкторе инициализируется GameMode для получения информации о доске и для перемещения фигур.

Наследуемый метод mousePressEvent считывает нажатие и вызывает метод класса GameMode для получения возможных ходов, далее подсвечивает элементы по указанным позициям, если встречается фигура, то цвет становится красным.

Наследуемый метод mouseReleaseEvent отслеживает отпускание клавиши мыши, которая передвигает фигуру на доске экземпляра GameMode и выполняет перемещение изображения фигуры.

```
class ChessScene : public QGraphicsScene {
    Q_OBJECT

public:
    ChessScene(int sceneSize, GameMode* gameMode, QObject *parent = nullptr) :
QGraphicsScene(parent);

    ~ChessScene();

private:
    std::vector<HighliteGraphicItem*> *highlitesGraphicItems;
    std::vector<QPoint> *allowedPoses;
    std::vector<std::vector<FigureGraphicItem*>> *figuresGraphicItems;
    GameMode *gameMode;
    int sceneSize;
    int sideSize;
    QPoint startPos;
    static const int SIDE_SIZE_IN_CELL = 8;

protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event) override;
    void mouseReleaseEvent(QGraphicsSceneMouseEvent *event) override;
    void removeHighlite();
    void addHighlite();

signals:
    void signalOtherTurn();
    void signalNewLog(QString log);
};
```

Листинг 6. Класс ChessScene.

3.1.2 Класс GameWindow

Класс GameWindow отрисовывает элементы игры и прошедшее время с начала первого хода. Для отсчёта времени используются сигналы и слоты из библиотеки QT. В конструкторе также устанавливается сцена для отображения доски с фигурами и логгер, который обрабатывается через слот. По истечению таймера высвечивается диалоговое сообщение, сигнализирующее о победе одного из игроков.

```
class GameWindow : public QWidget
{
    Q_OBJECT

public:
    explicit GameWindow(QWidget *parent = nullptr);
    GameWindow(GameMode *gameMode, QWidget *parent = nullptr);
    ~GameWindow();
    void keyPressEvent(QKeyEvent *event) override;

private:
    Ui::GameWindow *ui;
    QTimer timerOne;
    QTimer timerTwo;
    GameMode *gameMode;
    QTime timePlayerOne;
    QTime timePlayerTwo;
    QString timerToString(QTime &time);

private slots:
    void startTimerOne();
    void startTimerTwo();
    void decrementTimeOne();
    void decrementTimeTwo();
    void switchTimer();
    void postNewLog(QString event);
};
```

Листинг 7. Интерфейс класса GameWindow.

Ниже представлен пример отображения (Рисунок 2). Сообщения о движениях фигур формируются в классе GameMode и через сигнал в классе ChessScene попадает на экран.

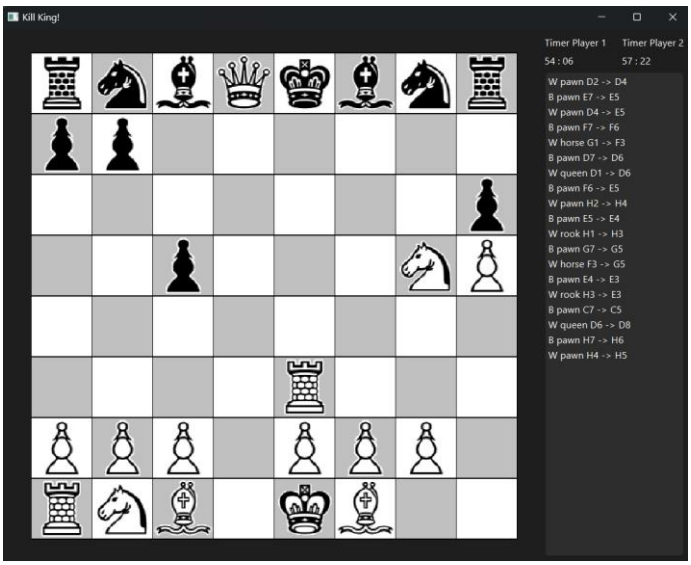

Рисунок 2. Демонстрация отрисовки окна игры.

3.2 Элементы меню

Все окна представляют собой виджеты, которые хранятся и отображаются через WindowHolder. WindowHolder – это класс-синглтон, который имеет в себе стэк с отображенными ранее окнами. Это позволяет возвращаться обратно по кнопке.

```
class WindowHolder
{
public:
    static QMainWindow* instance(QWidget *widget);
    static void holdPrevWidget();

private:
    Ui::WindowHolder *ui;
    WindowHolder();
    ~WindowHolder();
    inline static QMainWindow *windowHolder = nullptr;
    inline static QStackedWidget *widgetStack = nullptr;
    WindowHolder(WindowHolder const&);
    WindowHolder& operator= (WindowHolder const&);
};
```

Листинг 8. Класс WindowHolder.

Остальные элементы меню представляет собой меню с несколькими кнопками, которые переносят на другие окна.

4. Настройки

4.1 Класс SettingsJson

Класс имеет поля, которые можно инициализировать или сохранить с помощью статических методов. Сохранение производится в json файл.

```
class SettingsJson {
public:
    static bool saveParametresToJsonFile(SettingsJson settingsJson);
    static bool updateParametresFromJsonFile(SettingsJson &settingsJson);

    FigureColor colorFigureInBottom = FigureColor::WHITE;
    int minOnTimer = 15;
    int secOnTimer = 30;
    FigureDisposition figureDisposition = FigureDisposition::CLASSIC;
    std::string fileNameOfFigureDisposition = "";

private:
    static bool saveDefaulteParameter(SettingsJson& settingsJson);
    static constexpr char FILE_PATH_TO_JSON[] = "settings.json";
};
```

Листинг 9. Класс SettingsJson.

4.2 Класс SettingsWindow

Класс представляет пользовательский интерфейс для формирования и сохранения настроек. Кнопка «Сохранить» выполняет обновление json файла полями экземпляра SettingsJson.

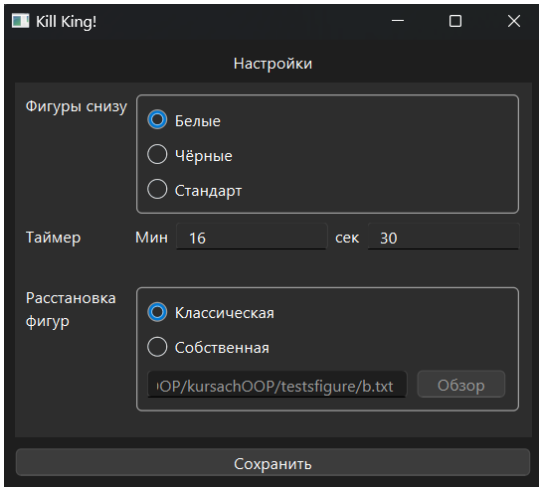

Рисунок 3. Пользовательский интерфейс настроек.

Таймер задаётся и используется в зависимости от выбранного режима игры, то есть в онлайн она будет задаваться хостом.

Расстановка фигур позволяет загрузить собственное поле и получить собственную игровую доску.

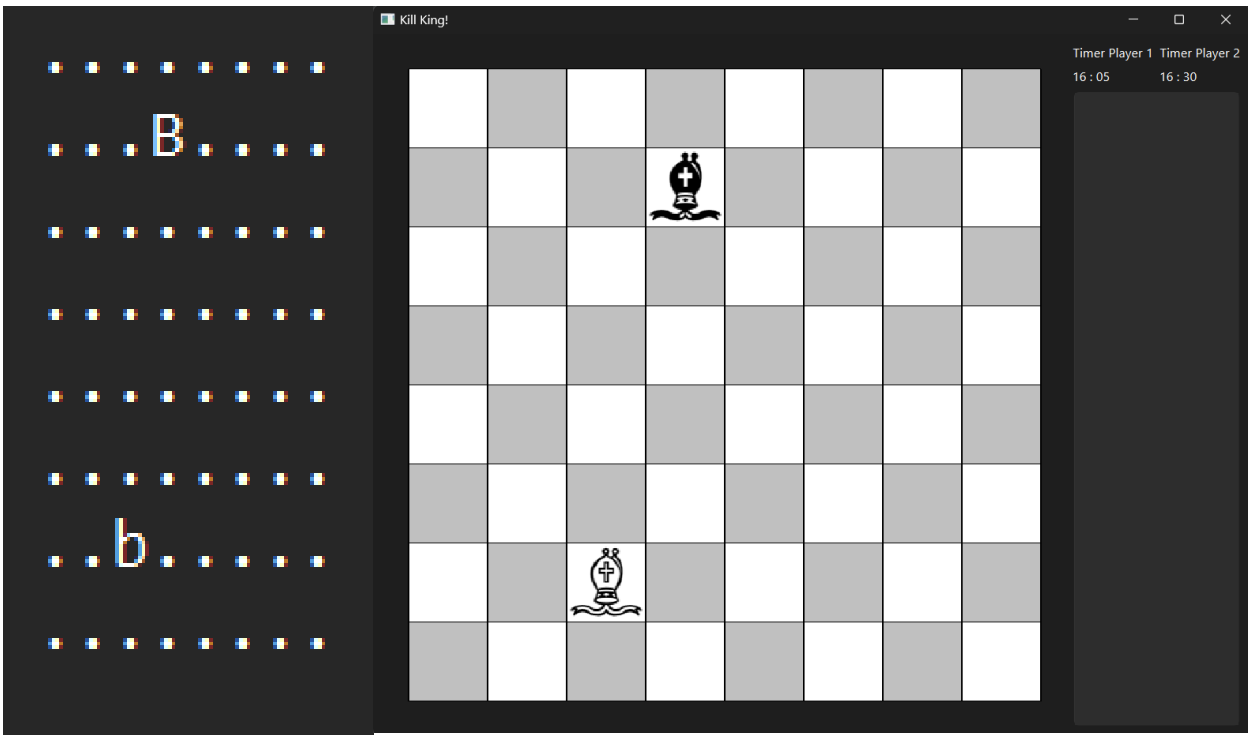

Рисунок 4. Демонстрация, собственная игровая доска.

Заключение

В ходе курсовой работы был разработан шахматный движок и пользовательский интерфейс под него. Был применен json файлы для реализации настроек. Для пользовательского интерфейса применена библиотека QT, механизм сигналов и слотов позволил связать некоторые узлы программы и реализовать обработку действий пользователя. Для достижения гибкости и расширяемости применены паттерны «Одиночка» и «Стратегия». Расширены навыки владения языка программирования C++. Укреплены знания создания UML-диаграмм и объектной декомпозиции.

Список источников и литературы

1. Receive Drag/Drop events for items in QGraphicsScene [Электронный ресурс] Режим доступа: https://forum.qt.io/topic/128501/receive-drag-drop-events-for-items-in-qgraphicsscene/5
2. Bitboards [Электронный ресурс] Режим доступа: https://www.chessprogramming.org/Bitboards#Pattern_and_Attacks
3. Шпаргалка по шаблонам проектирования [Электронный ресурс] Режим доступа: https://habr.com/ru/articles/210288/
4. Синглтон на C++ [Электронный ресурс] Режим доступа: https://www.modernescpp.com/index.php/creational-patterns-singleton/
5. QT doc QWidget [Электронный ресурс] Режим доступа: https://doc.qt.io/qt-6/qwidget.html
6. JSON for Modern C++ [Электронный ресурс] Режим доступа: https://github.com/nlohmann/json
7. Сохранение данных в файл средствами Qt [Электронный ресурс] Режим доступа: https://itnotesblog.ru/note/sohranenie-dannyh-v-fajl-sredstvami-qt
8. Dumb7Fill [Электронный ресурс] Режим доступа: https://www.chessprogramming.org/Dumb7Fill
9. Шпаргалка по UML [Электронный ресурс] Режим доступа: https://jenyay.net/Programming/Uml

Приложения

Код:

actionbehavior.h

```
#ifndef ACTIONBEHAVIOR_H
#define ACTIONBEHAVIOR_H

#include <stdint.h>
#include <QtLogging>
#include <functional>

class ActionBehavior {
public:
    virtual uint64_t action(uint64_t bitboard, uint64_t blocking) {
        return 0ULL;
    }

protected:
    static const uint64_t RANK_8 = 0xFF00000000000000;
    static const uint64_t RANK_7 = 0x00FF000000000000;
    static const uint64_t RANK_6 = 0x0000FF0000000000;
    static const uint64_t RANK_5 = 0x000000FF00000000;
    static const uint64_t RANK_4 = 0x00000000FF000000;
    static const uint64_t RANK_3 = 0x0000000000FF0000;
    static const uint64_t RANK_2 = 0x000000000000FF00;
    static const uint64_t RANK_1 = 0x00000000000000FF;
    static const uint64_t FILE_A = 0x0101010101010101;
    static const uint64_t FILE_B = 0x0202020202020202;
    static const uint64_t FILE_H = 0x8080808080808080;
    static const uint64_t FILE_G = 0x7070707070707070;

    static const uint64_t EMPTY_MASK = 0ULL;

    static uint64_t genRay(
        std::function<uint64_t(uint64_t)> bitShift,
        uint64_t file,
        uint64_t bitboard,
        uint64_t blocking) {
        uint64_t flood = bitboard;
        uint64_t empty = ~blocking & ~file;
        flood |= bitboard = (bitShift(bitboard)) & empty;
        flood |= bitboard = (bitShift(bitboard)) & empty;
        flood |= bitboard = (bitShift(bitboard)) & empty;
        flood |= bitboard = (bitShift(bitboard)) & empty;
        flood |= bitboard = (bitShift(bitboard)) & empty;
        flood |=            (bitShift(bitboard)) & empty;
        return              (bitShift(flood)) & ~file;
    }
};

#endif // ACTIONBEHAVIOR_H
```

alldirectionsaction.h

```
#ifndef ALLDIRECTIONSACTION_H
#define ALLDIRECTIONSACTION_H

#include "actionbehavior.h"
```

```
class AllDirectionsBehavior : public ActionBehavior {
public:
    uint64_t action(uint64_t bitboard, uint64_t blocking) override {
        return ((bitboard << 7) | (bitboard >> 9) | (bitboard >> 1)) & ~FILE_H
               | ((bitboard >> 7) | (bitboard << 9) | (bitboard << 1)) & ~FILE_A
               | (bitboard >> 8) | bitboard << 8;
    }
};

#endif // ALLDIRECTIONSACTION_H
```

allslidedirectionaction.h

```
#ifndef ALLSLIDEDIRECTIONACTION_H
#define ALLSLIDEDIRECTIONACTION_H

#include "actionbehavior.h"
#include "diagonalaction.h"
#include "straightaction.h"

class AllSlideDirectionAction : public ActionBehavior {
public:
    uint64_t action(uint64_t bitboard, uint64_t blocking) {
        return DiagonalAction().action(bitboard, blocking)
               | StraightAction().action(bitboard, blocking);
    }
};

#endif // ALLSLIDEDIRECTIONACTION_H
```

diagonalaction.h

```
#ifndef DIAGONALACTION_H
#define DIAGONALACTION_H

#include "actionbehavior.h"

class DiagonalAction : public ActionBehavior {
public:
    uint64_t action(uint64_t bitboard, uint64_t blocking) {
        return genRay([](uint64_t x){return x << 7;},
                      FILE_H, bitboard, blocking)
               | genRay([](uint64_t x){return x >> 7;},
                        FILE_A, bitboard, blocking)
               | genRay([](uint64_t x){return x >> 9;},
                        FILE_H, bitboard, blocking)
               | genRay([](uint64_t x){return x << 9;},
                        FILE_A, bitboard, blocking);
    }
};

#endif // DIAGONALACTION_H
```

horseaction.h

```
#ifndef HORSEACTION_H
#define HORSEACTION_H

#include "actionbehavior.h"

class HorseAction : public ActionBehavior {
public:
```

```
    uint64_t action(uint64_t bitboard, uint64_t blocking) override {
        const uint64_t notAFile = ~FILE_A;
        const uint64_t notABFile = notAFile & ~FILE_B;
        const uint64_t notHFile = ~FILE_H;
        const uint64_t notGHFile = notHFile & ~FILE_G;
        return (bitboard << 17) & notAFile | (bitboard << 10) & notABFile |
(bitboard >> 6) & notABFile | (bitboard >> 15) & notAFile |
               (bitboard << 15) & notHFile | (bitboard << 6) & notGHFile |
(bitboard >> 10) & notGHFile | (bitboard >> 17) & notHFile;
    }
};

#endif // HORSEACTION_H
```

noaction.h

```
#ifndef NOACTION_H
#define NOACTION_H

#include "actionbehavior.h"

class NoAction : public ActionBehavior {
public:
    uint64_t action(uint64_t bitboard, uint64_t blocking) override {
        return 0;
    }
};

#endif // NOACTION_H
```

pawnaction.h

```
#ifndef PAWNACTION_H
#define PAWNACTION_H

#include "actionbehavior.h"

class PawnWhiteMove : public ActionBehavior {
public:
    uint64_t action(uint64_t bitboard, uint64_t blocking) override {
        return (bitboard << 8) | (((bitboard & RANK_2) << 16));
                                  // & ~(blocking | (blocking & ~RANK_2)<< 8));
    }
};

class PawnWhiteAttack : public ActionBehavior {
public:
    uint64_t action(uint64_t bitboard, uint64_t blocking) override {
        return ((bitboard << 9) & ~FILE_A | (bitboard << 7) & (~FILE_H)) &
blocking;
    }
};

class PawnBlackMove : public ActionBehavior {
public:
    uint64_t action(uint64_t bitboard, uint64_t blocking) override {
        return ((bitboard >> 8) | (bitboard & RANK_7) >> 16);
               // & ~(blocking | (blocking & ~RANK_7) >> 8);;
    }
};

class PawnBlackAttack : public ActionBehavior {
```

```
public:
    uint64_t action(uint64_t bitboard, uint64_t blocking) override {
        return ((bitboard >> 9) & ~FILE_H | (bitboard >> 7) & (~FILE_A)) &
blocking;
    }
};

#endif // PAWNACTION_H
```

straightaction.h

```
#ifndef STRAIGHTACTION_H
#define STRAIGHTACTION_H

#include "actionbehavior.h"

class StraightAction : public ActionBehavior {
public:
    uint64_t action(uint64_t bitboard, uint64_t blocking) {
        return genRay([](uint64_t x){return x << 1;},
                      FILE_A, bitboard, blocking)
               | genRay([](uint64_t x){return x >> 1;},
                      FILE_H, bitboard, blocking)
               | genRay([](uint64_t x){return x >> 8;},
                        EMPTY_MASK, bitboard, blocking)
               | genRay([](uint64_t x){return x << 8;},
                        EMPTY_MASK, bitboard, blocking);
    }
};

#endif // STRAIGHTACTION_H
```

bishop.h

```
#ifndef BISHOP_H
#define BISHOP_H

#include "figure.h"
#include "behavior/diagonalaction.h"

#include <QCoreApplication>

class Bishop : public Figure {
public:
    Bishop(FigureColor figureColor)
        : Figure(figureColor, "bishop", new DiagonalAction()) {
        setPathToImage(figureColor, "B");
    }
};

#endif // BISHOP_H
```

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

#include "behavior/actionbehavior.h"
#include "../enums.h"
#include <cstdio>
#include <string>
#include <QString>
#include <QCoreApplication>
```

```
class Figure {
public:
    bool const moveEqualAttack;
    FigureColor const figureColor;
    QString const name;

    Figure(
            FigureColor figureColor,
            QString name,
            ActionBehavior *moveWithAttackBehavior
        ) : moveEqualAttack(true),
            figureColor(figureColor),
            name(name),
            movePerform(moveWithAttackBehavior),
            attackPerform(moveWithAttackBehavior) {
        filePathToImage = filePathToImage;
    }

    Figure(
            FigureColor figureColor,
            QString name,
            ActionBehavior *moveBehavior,
            ActionBehavior *attackBehavior
        ) : moveEqualAttack(false),
            figureColor(figureColor),
            name(name),
            movePerform(moveBehavior),
            attackPerform(attackBehavior) {
        filePathToImage = filePathToImage;
    }

    uint64_t move(uint64_t bitboard, uint64_t blocking) {
        return movePerform->action(bitboard, blocking);
    }

    uint64_t attack(uint64_t bitboard, uint64_t blocking) {
        return attackPerform->action(bitboard, blocking);
    }

    ~Figure() {
        delete movePerform;
        if (!moveEqualAttack)
            delete attackPerform;
    }

    QString getFilePathToImage() {
        return filePathToImage;
    }

protected:
    QString filePathToImage;

    void setPathToImage(FigureColor figureColor, const char* name) {
        QString res;
        char colorDir[7];
        if (figureColor == FigureColor::WHITE)
            sprintf(colorDir, "white/");
        else if (figureColor == FigureColor::BLACK)
            sprintf(colorDir, "black/");
        else
```

```
            return;

        filePathToImage = QString()
                                .append(":/").append("resources/").append(colorDir)
                                .append(name).append(".png");
    }

private:
    ActionBehavior *movePerform;
    ActionBehavior *attackPerform;
};

#endif // FIGURE_H
```

horse.h

```
#ifndef HORSE_H
#define HORSE_H

#include "figure.h"
#include "behavior/horseaction.h"

class Horse : public Figure {
public:
    Horse(FigureColor figureColor)
        : Figure(figureColor, "horse", new HorseAction()) {
        setPathToImage(figureColor, "H");
    }
};

#endif // HORSE_H
```

king.h

```
#ifndef KING_H
#define KING_H

#include "figure.h"
#include "behavior/alldirectionsaction.h"

class King : public Figure {
public:
    King(FigureColor figureColor)
        : Figure(figureColor, "king", new AllDirectionsBehavior()) {
        setPathToImage(figureColor, "K");
    }
};

#endif // KING_H
```

nonefigure.h

```
#ifndef NONEFIGURE_H
#define NONEFIGURE_H

#include "figure.h"
#include "behavior/noaction.h"

class NoneFigure : public Figure {
public:
    NoneFigure() : Figure(FigureColor::DEFAULT, "", new NoAction()) {}
};
```

```
#endif // NONEFIGURE_H
```

pawn.h

```
#ifndef PAWN_H
#define PAWN_H

#include "figure.h"
#include "behavior/pawnaction.h"

class PawnWhite : public Figure {
public:
    PawnWhite()
        : Figure(FigureColor::WHITE, "pawn", new PawnWhiteMove(), new
PawnWhiteAttack()) {
        setPathToImage(figureColor, "P");
    }
};

class PawnBlack : public Figure {
public:
    PawnBlack()
        : Figure(FigureColor::BLACK, "pawn", new PawnBlackMove(), new
PawnBlackAttack()) {
        setPathToImage(figureColor, "P");
    }
};

#endif // PAWN_H
```

queen.h

```
#ifndef QUEEN_H
#define QUEEN_H

#include "figure.h"
#include "behavior/allslidedirectionaction.h"

class Queen : public Figure {
public:
    Queen(FigureColor figureColor)
        : Figure(figureColor, "queen", new AllSlideDirectionAction()) {
        setPathToImage(figureColor, "Q");
    }
};

#endif // QUEEN_H
```

rook.h

```
#ifndef ROOK_H
#define ROOK_H

#include "figure.h"
#include "behavior/straightaction.h"

class Rook : public Figure {
public:
    Rook(FigureColor figureColor)
        : Figure(figureColor, "rook", new StraightAction()) {
        setPathToImage(figureColor, "R");
    }
};
```

```
#endif // ROOK_H
```

chesslogger.h

```
#ifndef CHESSLOGGER_H
#define CHESSLOGGER_H

#include <QObject>
#include <QPoint>
#include <QString>
#include <QDebug>

#include "../figure/figure.h"


class ChessLogger {
public:
    static QString moving(QPoint pos, QPoint target) {
        return QString()
                .append(" ")
                .append(namedCalls[pos.y()][pos.x()])
                .append(" -> ")
                .append(namedCalls[target.y()][target.x()]);
    }

    static QString moving(QPoint pos, QPoint target, Figure *figure) {
        return QString()
            .append(figure->figureColor == FigureColor::WHITE ? "W " : "B ")
            .append(figure->name)
            .append(" ")
            .append(namedCalls[pos.y()][pos.x()])
            .append(" -> ")
            .append(namedCalls[target.y()][target.x()]);
    }

private:
    inline static std::vector<std::vector<const char*>> namedCalls {
        {"A1", "B1", "C1", "D1", "E1", "F1", "G1", "H1"},
        {"A2", "B2", "C2", "D2", "E2", "F2", "G2", "H2"},
        {"A3", "B3", "C3", "D3", "E3", "F3", "G3", "H3"},
        {"A4", "B4", "C4", "D4", "E4", "F4", "G4", "H4"},
        {"A5", "B5", "C5", "D5", "E5", "F5", "G5", "H5"},
        {"A6", "B6", "C6", "D6", "E6", "F6", "G6", "H6"},
        {"A7", "B7", "C7", "D7", "E7", "F7", "G7", "H7"},
        {"A8", "B8", "C8", "D8", "E8", "F8", "G8", "H8"}
    };
};

#endif // CHESSLOGGER_H
```

gamemode.h

```
#ifndef GAMEMODE_H
#define GAMEMODE_H

#include <string>
#include <vector>
#include <set>
#include <algorithm>
#include <functional>
#include <QPoint>
```

```
#include <QTimer>
#include <QTime>

#include "../figure/figure.h"
#include "../figure/nonefigure.h"
#include "../figure/king.h"
#include "../figure/bishop.h"
#include "../figure/horse.h"
#include "../figure/pawn.h"
#include "../figure/rook.h"
#include "../figure/queen.h"
#include "../settings/settingsjson.h"
#include "../utils.h"
#include "../figure/behavior/actionbehavior.h"

#include "chesslogger.h"

class GameMode {
public:
    GameMode() {
        SettingsJson::updateParametresFromJsonFile(settings);
    }

    ~GameMode() {

    }

    bool isConteinFigureByPos(QPoint realPos) {
        QPoint pos = QPoint(realPos.x(), invCoord(realPos.y()));
        return figurePlacement[pos.y()][pos.x()];
    }

    std::vector<std::vector<Figure*>> getFigurePlacementInv() {
        return figurePlacement;
    }

    virtual std::vector<QPoint> *checkFigure(QPoint realPos) {
        QPoint pos = QPoint(realPos.x(), invCoord(realPos.y()));

        qDebug() << realPos << "by click" << pos << "for bitboard";

        Figure *figure = figurePlacement[pos.y()][pos.x()];
        if (figure != nullptr && figure->figureColor == turnColor) {
            std::vector<QPoint> *res
                = getPosesWithoutAlly(pos, [figure](uint64_t bb, uint64_t
bl){return figure->move(bb,bl);});

            if (!figure->moveEqualAttack) {
                std::vector<QPoint> *buf
                    = getPosesWithoutAlly(pos, [figure](uint64_t bb, uint64_t
bl){return figure->attack(bb,bl);});
                res->insert(res->begin(), buf->begin(), buf->end());
            }

            return res;
        }

        return new std::vector<QPoint>();
    }

    virtual bool moveFigure(QPoint realPos, QPoint realTarget) {
```

```
        qDebug() << "realPos:" << realPos << realTarget;

        QPoint pos = QPoint(realPos.x(), invCoord(realPos.y()));
        QPoint target = QPoint(realTarget.x(), invCoord(realTarget.y()));

        qDebug() << "pos:" << pos << target;
        if (figurePlacement[pos.y()][pos.x()]->figureColor == turnColor) {
            logMsg = ChessLogger::moving(pos, target,
figurePlacement[pos.y()][pos.x()]);

            Figure *temp = figurePlacement[target.y()][target.x()];
            if (temp != nullptr) delete temp;
            figurePlacement[target.y()][target.x()] =
figurePlacement[pos.y()][pos.x()];
            if (figurePlacement[pos.y()][pos.x()] != nullptr)
figurePlacement[pos.y()][pos.x()] = nullptr;

            figureDispositionBitboard = figureDispositionBitboard &
~convertPosToBitboard(pos) | convertPosToBitboard(target);

            return true;
        }

        return false;
    }

    QString getNewLogMsg() {
        return logMsg;
    }

    virtual void switchTurn() { qWarning() << "switchTurn not realized"; }

    static int invCoord(int coord) {
        return BOARD_SIDE - coord - 1;
    }

    QTime timeLeftPlayerOne;
    QTime timeLeftPlayerTwo;

protected:
    QString logMsg;

    SettingsJson settings;

    std::vector<std::vector<Figure*>> figurePlacement;

    FigureColor turnColor = FigureColor::WHITE;

    uint64_t figureDispositionBitboard;

    std::vector<std::vector<Figure*>> convertFileFromSettingsToDisposition() {
        QFile in(Utils::toQString(settings.fileNameOfFigureDisposition));
        if (in.open(QIODevice::ReadOnly)) {
            QTextStream stream(&in);

            std::string customDisposition = stream.readAll().toStdString();

            in.close();
            return convertStringToDisposition(customDisposition);
        }
```

```
        return convertStringToDisposition(classicDisposition);

    }

    std::vector<QPoint> *getPosesWithoutAlly(QPoint& pos,
std::function<uint64_t(uint64_t, uint64_t)> action) {
        std::vector<QPoint> *resNotProcessed = convertBitboardToPoses(
            action(
                convertPosToBitboard(pos),
                figureDispositionBitboard
                )
            );

        qDebug() << convertPosToBitboard(pos);
        for (auto it : *resNotProcessed) {
            qDebug() << it;
        }

        std::vector<QPoint> *res = new std::vector<QPoint>;
        for (QPoint it : *resNotProcessed) {
            if (this->figurePlacement[it.y()][it.x()] == nullptr
                || this->figurePlacement[it.y()][it.x()]->figureColor != this-
>turnColor)
                res->push_back(it);
        }

        delete resNotProcessed;

        for (QPoint &it : *res) {
            it.setY(invCoord(it.y()));
            qDebug() << it;
        }

        return res;
    }

    static const int BOARD_SIDE = 8;

    static std::vector<std::vector<Figure*>>
convertStringToDisposition(std::string disposition) {
        std::vector<std::vector<Figure*>> res(BOARD_SIDE,
std::vector<Figure*>(BOARD_SIDE));
        int shift = 0; // for protection from other symbols
        int iterWrite = 0;
        int iterRead = 0;
        while (iterRead < BOARD_SIDE * BOARD_SIDE + shift) {
            Figure *figure = nullptr;
            char c = disposition[iterRead];
            switch (toupper(c)) {
            case 'R':
                figure = new Rook(defineColorByChar(c));
                break;
            case 'H':
                figure = new Horse(defineColorByChar(c));
                break;
            case 'B':
                figure = new Bishop(defineColorByChar(c));
                break;
            case 'Q':
                figure = new Queen(defineColorByChar(c));
                break;
```

```
            case 'K':
                figure = new King(defineColorByChar(c));
                break;
            case 'P':
                figure = defineColorByChar(c) == FigureColor::WHITE
                            ? static_cast<Figure*>(new PawnWhite)
                            : static_cast<Figure*>(new PawnBlack);
                break;
            // case 'A': // assassin - horse att bishop move
            //     break;

            // case 'S': // spearman - rook att king move
            //     break;
            case '.':
                iterWrite++;
                break;
            default:
                shift++;
                break;
            }

            if (figure != nullptr) {
                res[invCoord(iterWrite / BOARD_SIDE)][iterWrite % BOARD_SIDE] =
figure;
                iterWrite++;
            }

            iterRead++;
        }

        return res;
    }

    inline static const std::string classicDisposition =
        "RHBQKBHR\
        PPPPPPPP\
        ........\
        ........\
        ........\
        ........\
        pppppppp\
        rhbqkbhr";

    static FigureColor defineColorByChar(int c) {
        return islower(c) ? FigureColor::WHITE : FigureColor::BLACK;
    }

    static uint64_t convertPosToBitboard(QPoint pos) {
        return (1ULL << (pos.y() * 8ULL + pos.x()));
    }

    static std::vector<QPoint>* convertBitboardToPoses(uint64_t bitboard) {
        std::vector<QPoint>* res = new std::vector<QPoint>;

        for (int i = 0; i < BOARD_SIDE * BOARD_SIDE; i++) {
            if ((bitboard >> i) & 1)
                res->push_back(QPoint(i % BOARD_SIDE, i / BOARD_SIDE));
        }

        return res;
    }
```

```
    static uint64_t convertDispositionToBitboard(std::vector<std::vector<Figure*>>
disposition) {
        uint64_t bitboard = 0ULL;

        for (int y = 0; y < 8; y++) {
            for (int x = 0; x < 8; x++) {
                Figure *figure = disposition[y][x];
                if (figure != nullptr) {
                    uint64_t square = 1ULL << (y * 8 + x);
                    bitboard |= square;
                }
            }
        }

        return bitboard;
    }
};

#endif // GAMEMODE_H
```

pvspoffline.h

```
#ifndef PVSPOFFLINE_H
#define PVSPOFFLINE_H

#include "gamemode.h"

class PVsPOffline : public GameMode {
public:
    PVsPOffline() : GameMode() {
        if (settings.figureDisposition == FigureDisposition::CUSTOM)
            figurePlacement = convertFileFromSettingsToDisposition();
        else
            figurePlacement = convertStringToDisposition(classicDisposition);

        figureDispositionBitboard = convertDispositionToBitboard(figurePlacement);

        timeLeftPlayerOne = QTime(0, settings.minOnTimer, settings.secOnTimer);
        timeLeftPlayerTwo = QTime(0, settings.minOnTimer, settings.secOnTimer);
    }

    void switchTurn() override {
        if (turnColor == FigureColor::WHITE)
            turnColor = FigureColor::BLACK;
        else if (turnColor == FigureColor::BLACK)
            turnColor = FigureColor::WHITE;
    }
};

#endif // PVSPOFFLINE_H
```

cellgraphicitem.h

```
#ifndef CELLGRAPHICITEM_H
#define CELLGRAPHICITEM_H

#include <QGraphicsView>

class CellGraphicItem : public QGraphicsRectItem {
public:
```

```
    CellGraphicItem(QPoint scenePos, int size, bool isWhite, QGraphicsItem* parent
= nullptr)
        : QGraphicsRectItem(scenePos.x(), scenePos.y(), size, size, parent) {
        setBrush(isWhite ? Qt::white : Qt::lightGray);
    }
};

#endif // CELLGRAPHICITEM_H
```

chessscene.h

```
#ifndef CHESSSCENE_H
#define CHESSSCENE_H

#include <QGraphicsScene>
#include <QGraphicsItem>

#include "cellgraphicitem.h"
#include "figuregraphicitem.h"
#include "highlitegraphicitem.h"

#include "../gamemode/chesslogger.h"

class ChessScene : public QGraphicsScene {
    Q_OBJECT

public:
    ChessScene(int sceneSize, GameMode* gameMode, QObject *parent = nullptr) :
QGraphicsScene(parent) {
        // cellsGraphicItems = new std::vector<CellGraphicItem*>;
        highlitesGraphicItems = new std::vector<HighliteGraphicItem*>;
        figuresGraphicItems = new
std::vector<std::vector<FigureGraphicItem*>>(SIDE_SIZE_IN_CELL,
std::vector<FigureGraphicItem*>(SIDE_SIZE_IN_CELL));
        allowedPoses = new std::vector<QPoint>;

        this->gameMode = gameMode;

        this->sceneSize = sceneSize;
        sideSize = sceneSize / SIDE_SIZE_IN_CELL;

        bool isWhite = true;
        for (int i = 0; i < sceneSize; i += sideSize) {
            for (int j = 0; j < sceneSize; j += sideSize) {
                CellGraphicItem* it = new CellGraphicItem(QPoint(j, i), sideSize,
isWhite);
                it->setZValue(0);
                addItem(it);
                isWhite = !isWhite;
            }
            isWhite = !isWhite;
        }

        for (int y = 0; y < SIDE_SIZE_IN_CELL; ++y) {
            for (int x = 0; x < SIDE_SIZE_IN_CELL; ++x) {
                Figure* figure = gameMode-
>getFigurePlacementInv()[GameMode::invCoord(y)][x];
                if (figure != nullptr) {
                    FigureGraphicItem *it
                        = new FigureGraphicItem(Utils::toScenePos(QPoint(x, y),
sideSize), sideSize);
```

```
                    (*figuresGraphicItems)[y][x] = it;
                    QString filePath = figure->getFilePathToImage();
                    it->setFigurePixmap(filePath);
                    it->setZValue(2);
                    addItem(it);
                }
            }
        }
    }

    ~ChessScene() {
        // delete cellsGraphicItems;
        delete highlitesGraphicItems;
        delete figuresGraphicItems;
    }



private:
    // std::vector<CellGraphicItem*> *cellsGraphicItems;
    std::vector<HighliteGraphicItem*> *highlitesGraphicItems;
    std::vector<QPoint> *allowedPoses;
    std::vector<std::vector<FigureGraphicItem*>> *figuresGraphicItems;

    GameMode *gameMode;

    int sceneSize;
    int sideSize;

    QPoint startPos;

    static const int SIDE_SIZE_IN_CELL = 8;

protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *event) override {
        removeHighlite();

        startPos = Utils::toPos(event->scenePos().toPoint(), sideSize);
        allowedPoses = gameMode->checkFigure(Utils::toPos(event-
>scenePos().toPoint(), sideSize));

        addHighlite();
    }

    void mouseReleaseEvent(QGraphicsSceneMouseEvent *event) override {
        for (QPoint endPos : *allowedPoses)
            if (endPos == Utils::toPos(event->scenePos().toPoint(), sideSize)) {
                gameMode->moveFigure(startPos, endPos);
                gameMode->switchTurn();

                FigureGraphicItem *temp =
(*figuresGraphicItems)[endPos.y()][endPos.x()];
                if(temp != nullptr) delete temp;
                (*figuresGraphicItems)[endPos.y()][endPos.x()] =
(*figuresGraphicItems)[startPos.y()][startPos.x()];
                (*figuresGraphicItems)[endPos.y()][endPos.x()]-
>setPos(Utils::toScenePos(endPos, sideSize));
                if((*figuresGraphicItems)[startPos.y()][startPos.x()] != nullptr)
(*figuresGraphicItems)[startPos.y()][startPos.x()] = nullptr;

                emit signalNewLog(gameMode->getNewLogMsg());
```

```
                emit signalOtherTurn();

                break;
            }

        removeHighlite();
    }

    void removeHighlite() {
        allowedPoses->clear();
        for (HighliteGraphicItem* it : *highlitesGraphicItems)
            removeItem(it);
        highlitesGraphicItems->clear();
    }

    void addHighlite() {
        for (QPoint it : *allowedPoses) {
            HighliteGraphicItem* newHighliteGraphicItem
                = new HighliteGraphicItem(Utils::toScenePos(it, sideSize),
sideSize, !gameMode->isConteinFigureByPos(it));
            newHighliteGraphicItem->setZValue(1);

            highlitesGraphicItems->push_back(newHighliteGraphicItem);

            addItem(newHighliteGraphicItem);
        }
    }

signals:
    void signalOtherTurn();
    void signalNewLog(QString log);
};

#endif // CHESSSCENE_H
```

figuregraphicitem.h

```
#ifndef FIGUREGRAPHICITEM_H
#define FIGUREGRAPHICITEM_H

#include <QGraphicsItem>
#include <QGraphicsScene>
#include <QGraphicsSceneMouseEvent>
#include <vector>

#include "../gamemode/gamemode.h"
#include "../utils.h"

class FigureGraphicItem : public QObject, public QGraphicsPixmapItem {
    Q_OBJECT

public:
    FigureGraphicItem(QPoint scenePos, int size, const QString
fileNameToFigureImage, QGraphicsItem *parent = nullptr)
        : QGraphicsPixmapItem(parent) {
        this->size = size;
        setPos(scenePos.x(), scenePos.y());
        setFigurePixmap(fileNameToFigureImage);
    }

    FigureGraphicItem(QPoint scenePos, int size, QGraphicsItem *parent = nullptr)
        : QGraphicsPixmapItem(parent) {
```

```
        this->size = size;
        setPos(scenePos.x(), scenePos.y());
    }

    void setFigurePixmap(QString fileNameToFigureImage) {
        qDebug() << fileNameToFigureImage << "set pixmap";
        QImage image(fileNameToFigureImage);
        setPixmap(QPixmap::fromImage(image).scaled(size, size));
    }

private:
    int size;
};

#endif // FIGUREGRAPHICITEM_H
```

highlitegraphicitem.h

```
#ifndef HIGHLITEGRAPHICITEM_H
#define HIGHLITEGRAPHICITEM_H

#include <QGraphicsItem>

class HighliteGraphicItem : public QGraphicsRectItem {
public:
    HighliteGraphicItem(QPoint scenePos, int size, bool isEnemy, QGraphicsItem
*parent = nullptr)
        : QGraphicsRectItem(scenePos.x(), scenePos.y(), size, size, parent) {
        setBrush(isEnemy ? Qt::green : Qt::red);
    }
};

#endif // HIGHLITEGRAPHICITEM_H
```

settingsjson.h

```
#ifndef SETTINGSJSON_H
#define SETTINGSJSON_H

#include <QFile>
#include <QTextStream>
#include <string>

#include "../enums.h"
#include "../json.hpp"

using json = nlohmann::json;

class SettingsJson {
public:
    SettingsJson() {}
    ~SettingsJson() {}

    static bool saveParametresToJsonFile(SettingsJson settingsJson) {
        QFile out(FILE_PATH_TO_JSON);
        if (out.open(QIODevice::WriteOnly)) {
            QTextStream stream( &out );
            json jsonObj = {
                {"color_figure_in_bottom",
settingsJson.colorFigureInBottom},
                {"min_on_timer",                    settingsJson.minOnTimer},
                {"sec_on_timer",                    settingsJson.secOnTimer},
```

```
                {"figure_disposition",
settingsJson.figureDisposition},
                {"file_name_of_figure_disposition",
settingsJson.fileNameOfFigureDisposition}
            };

            qDebug() << jsonObj.dump().c_str();
            stream << jsonObj.dump().c_str();
            out.close();
            return true;
        }

        return false;
    }

    static bool updateParametresFromJsonFile(SettingsJson &settingsJson) {
        if (!QFile(FILE_PATH_TO_JSON).exists()) {
            qDebug() << "first launch, update defaulted values";
            saveDefaulteParameter(settingsJson);
            return true;
        }

        QFile in(FILE_PATH_TO_JSON);
        bool resultOfUpdate = false;
        if (in.open(QIODevice::ReadOnly)) {
            QTextStream stream( &in );
            try {
                json jsonObj = json::parse(stream.readAll().toStdString());

                settingsJson.colorFigureInBottom        =
jsonObj["color_figure_in_bottom"];
                settingsJson.minOnTimer                 =
jsonObj["min_on_timer"];
                settingsJson.secOnTimer                 =
jsonObj["sec_on_timer"];
                settingsJson.figureDisposition          =
jsonObj["figure_disposition"];
                settingsJson.fileNameOfFigureDisposition =
jsonObj["file_name_of_figure_disposition"];

                qDebug() << "updated values from json";

                resultOfUpdate = true;
            } catch (json::type_error e) {
                qDebug() << "type is not completable, update defaulted values";
                saveDefaulteParameter(settingsJson);
            } catch (json::parse_error e) {
                qDebug() << "parse incorrect, update defaulted values";
                saveDefaulteParameter(settingsJson);
            }

            in.close();
            return resultOfUpdate;
        } else {
            qDebug() << "file cannot open";
            return false;
        }
    }


    FigureColor colorFigureInBottom = FigureColor::WHITE;
```

```
    int minOnTimer = 15;
    int secOnTimer = 30;
    FigureDisposition figureDisposition = FigureDisposition::CLASSIC;
    std::string fileNameOfFigureDisposition = "";
    // ...

private:
    static bool saveDefaulteParameter(SettingsJson& settingsJson) {
        settingsJson = SettingsJson();
        return saveParametresToJsonFile(settingsJson);
    }

    static constexpr char FILE_PATH_TO_JSON[] = "settings.json";
};

#endif // SETTINGSJSON_H
```

gamemodesmenuwindow.h

```
#ifndef GAMEMODESMENUWINDOW_H
#define GAMEMODESMENUWINDOW_H

#include <QKeyEvent>
#include <QWidget>

namespace Ui {
class GameModesMenuWindow;
}

class GameModesMenuWindow : public QWidget
{
    Q_OBJECT

public:
    explicit GameModesMenuWindow(QWidget *parent = nullptr);
    ~GameModesMenuWindow();
    void keyPressEvent(QKeyEvent *event) override;

private slots:
    void on_btnPVsC_clicked();

    void on_btnPVsPOfflane_clicked();

    void on_btnPVsPOnline_clicked();

private:
    Ui::GameModesMenuWindow *ui;
};

#endif // GAMEMODESMENUWINDOW_H
```

gamewindow.h

```
#ifndef GAMEWINDOW_H
#define GAMEWINDOW_H

#include <QWidget>
#include "../gamemode/gamemode.h"
#include "../gamemode/chesslogger.h"

namespace Ui {
class GameWindow;
```

```
}

class GameWindow : public QWidget
{
    Q_OBJECT

public:
    explicit GameWindow(QWidget *parent = nullptr);
    GameWindow(GameMode *gameMode, QWidget *parent = nullptr);
    ~GameWindow();
    void keyPressEvent(QKeyEvent *event) override;

private:
    Ui::GameWindow *ui;
    QTimer timerOne;
    QTimer timerTwo;
    GameMode *gameMode;
    QTime timePlayerOne;
    QTime timePlayerTwo;

    QString timerToString(QTime &time);

private slots:
    void startTimerOne();
    void startTimerTwo();
    void decrementTimeOne();
    void decrementTimeTwo();
    void switchTimer();
    void postNewLog(QString event);
};

#endif // GAMEWINDOW_H
menuwindow.h
#ifndef MENUWINDOW_H
#define MENUWINDOW_H

#include <QKeyEvent>
#include <QWidget>

namespace Ui {
class MenuWindow;
}

class MenuWindow : public QWidget
{
    Q_OBJECT

public:
    explicit MenuWindow(QWidget *parent = nullptr);
    ~MenuWindow();
    void keyPressEvent(QKeyEvent *event) override;

private slots:
    void on_btnPlay_clicked();

    void on_btnSettings_clicked();

private:
    Ui::MenuWindow *ui;
};
```

```
#endif // MENUWINDOW_H
```

settingswindow.h

```
#ifndef SETTINGSWINDOW_H
#define SETTINGSWINDOW_H

#include <QWidget>
#include <QKeyEvent>
#include <QMainWindow>

#include "../settings/settingsjson.h"

namespace Ui {
class SettingsWindow;
}

class SettingsWindow : public QWidget
{
    Q_OBJECT

public:
    explicit SettingsWindow(QWidget *parent = nullptr);
    ~SettingsWindow();

    void keyPressEvent(QKeyEvent *event) override;

private slots:
    void on_btnSave_clicked();

    void on_btnSetFileNameToFigureDisposition_clicked();

    void on_rBtnClassicDisposition_clicked();

    void on_rBtnCustomDisposition_clicked();

private:
    Ui::SettingsWindow *ui;
};

#endif // SETTINGSWINDOW_H
```

windialog.h

```
#ifndef WINDIALOG_H
#define WINDIALOG_H

#include <QDialog>
#include <QHBoxLayout>
#include <QLabel>

class WinDialog : public QDialog {
public:
    WinDialog(int numberOfWinner, QWidget *parent = nullptr, Qt::WindowFlags f =
Qt::WindowFlags()) : QDialog(parent, f) {
        QBoxLayout* layout = new QHBoxLayout(parent);
        QLabel *label = new QLabel(parent);
        label->setText(QString().append("Игрок
").append(QString::number(numberOfWinner).append("Одержал победу!")));
        layout->addWidget(label);
    }
};
```

```
#endif // WINDIALOG_H
```

windowholder.h

```
#ifndef WINDOWHOLDER_H
#define WINDOWHOLDER_H

#include <QMainWindow>
#include <QHBoxLayout>
#include <QDebug>
#include <QStackedWidget>

namespace Ui {
class WindowHolder;
}

class WindowHolder
{
public:
    static QMainWindow* instance(QWidget *widget) {
        if (!windowHolder) {
            windowHolder = new QMainWindow();
            windowHolder->setWindowTitle("Kill King!");
            windowHolder->resize(400, 300);
            widget->setFocus();

            widgetStack = new QStackedWidget();
        }

        if (widget != nullptr) {
            widgetStack->addWidget(widget);
            widgetStack->setCurrentWidget(widget);
            windowHolder->setCentralWidget(widgetStack);

            qDebug() << "widget:" << widget << "is holded";
        }

        return windowHolder;
    }

    static void holdPrevWidget() {
        if(widgetStack == nullptr) {
            qDebug() << "widgetStack is null";
            return;
        }

        if (widgetStack->count()) {
            widgetStack->removeWidget(widgetStack->currentWidget());
        }

        if (widgetStack->count()) {
            windowHolder->setCentralWidget(widgetStack);
        }

        qDebug() << "hold prev widget, back to" << widgetStack->currentWidget();
    }

private:
    Ui::WindowHolder *ui;
    WindowHolder() {}
    ~WindowHolder() {}
```

```
    inline static QMainWindow *windowHolder = nullptr;
    inline static QStackedWidget *widgetStack = nullptr;

    WindowHolder(WindowHolder const&);
    WindowHolder& operator= (WindowHolder const&);
};

#endif // WINDOWHOLDER_H
```

enums.h

```
#ifndef ENUMS_H
#define ENUMS_H

enum FigureColor {
    WHITE,
    BLACK,
    DEFAULT
};

enum FigureDisposition {
    CLASSIC,
    CUSTOM
};

#endif // ENUMS_H
```

utils.h

```
#ifndef UTILS_H
#define UTILS_H

#include <string>
#include <QString>
#include <QPoint>

class Utils {
public:
    static const char* toCharArr(int x) {
        return std::to_string(x).c_str();
    }

    static QString toQString(std::string str) {
        return QString(str.c_str());
    }

    static QPoint toScenePos(QPoint pos, int sideSize) {
        return QPoint(pos.x() * sideSize, pos.y() * sideSize);
    }

    static QPoint toPos(QPoint scenePos, int sideSize) {
        return QPoint(scenePos.x() / sideSize, scenePos.y() / sideSize);
    }

    // static QString toQString(int x) {
    //     return QString(toCharArr(x));
    // }
};

#endif // UTILS_H
```

gamemodesmenuwindow.cpp

```
#include "gamemodesmenuwindow.h"
#include "ui_gamemodesmenuwindow.h"

#include "windowholder.h"
#include "gamewindow.h"
#include "../gamemode/pvspoffline.h"

GameModesMenuWindow::GameModesMenuWindow(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::GameModesMenuWindow)
{
    ui->setupUi(this);
}

GameModesMenuWindow::~GameModesMenuWindow()
{
    delete ui;
}

void GameModesMenuWindow::on_btnPVsC_clicked()
{
    // todo in next updates
}


void GameModesMenuWindow::on_btnPVsPOfflane_clicked()
{
    WindowHolder::instance(new GameWindow(new PVsPOffline()))->show();
    qDebug("type P vs P offline showing");
}


void GameModesMenuWindow::on_btnPVsPOnline_clicked()
{
    // todo in next updates
}

void GameModesMenuWindow::keyPressEvent(QKeyEvent *event) {
    if (event->key() == Qt::Key_Escape) {
        WindowHolder::holdPrevWidget();

        qDebug() << "escape pressed from" << this;
    }
}
```

gamewindow.cpp

```
#include "chessscene.h"
#include "gamewindow.h"
#include "ui_gamewindow.h"
#include "windialog.h"
#include "windowholder.h"

#include <QKeyEvent>

GameWindow::GameWindow(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::GameWindow)
{
    ui->setupUi(this);
}
```

```
GameWindow::GameWindow(GameMode *gameMode, QWidget *parent)
    : GameWindow::GameWindow(parent) {
    ChessScene *chessScene = new ChessScene(640, gameMode, this);
    this->gameMode = gameMode;

    this->timePlayerOne = gameMode->timeLeftPlayerOne;
    this->timePlayerTwo = gameMode->timeLeftPlayerTwo;

    ui->graphicsView->setScene(chessScene);

    ui->tLabelTimerPlayerOne->setText(timerToString(timePlayerOne));
    ui->tLabelTimerPlayerTwo->setText(timerToString(timePlayerTwo));

    startTimerOne();

    connect(chessScene, SIGNAL(signalOtherTurn()), this, SLOT(switchTimer()));

    connect(&timerOne, SIGNAL(timeout()), this, SLOT(decrementTimeOne()));
    connect(&timerTwo, SIGNAL(timeout()), this, SLOT(decrementTimeTwo()));

    connect(chessScene, SIGNAL(signalNewLog(QString)), this,
SLOT(postNewLog(QString)));
}

GameWindow::~GameWindow()
{
    delete ui;
}

void GameWindow::keyPressEvent(QKeyEvent *event) {
    if (event->key() == Qt::Key_Escape) {
        WindowHolder::holdPrevWidget();

        qDebug() << "escape pressed from" << this;
    }
}

void GameWindow::startTimerOne() {
    timerOne.start(1000);
}

void GameWindow::startTimerTwo() {
    timerTwo.start(1000);
}

void GameWindow::decrementTimeOne() {
    ui->tLabelTimerPlayerOne->setText(timerToString(timePlayerOne));
    timePlayerOne = timePlayerOne.addMSecs(-1000);

    if (timePlayerOne.hour() > 0) {
        WinDialog(2, this).show();
    }
}

void GameWindow::decrementTimeTwo() {
    ui->tLabelTimerPlayerTwo->setText(timerToString(timePlayerTwo));
    timePlayerTwo = timePlayerTwo.addMSecs(-1000);

    if (timePlayerOne.hour() > 0) {
        WinDialog(1, this).show();
    }
```

```
}

QString GameWindow::timerToString(QTime &time) {
    return QString(time.toString("mm : ss"));
}

void GameWindow::switchTimer() {
    qDebug() << "timer switched";

    if (timerOne.isActive()){
        timerOne.stop();
        timerTwo.start(1000);
    } else {
        timerTwo.stop();
        timerOne.start(1000);
    }

}

void GameWindow::postNewLog(QString event){
    ui->listWidget->addItem(event);
}
```

menuwindow.cpp

```
#include "menuwindow.h"
#include "ui_menuwindow.h"
#include "windowholder.h"
#include "settingswindow.h"
#include "gamemodesmenuwindow.h"

#include <QMainWindow>
#include <QtLogging>

MenuWindow::MenuWindow(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::MenuWindow)
{
    ui->setupUi(this);
}

MenuWindow::~MenuWindow()
{
    delete ui;
}

void MenuWindow::on_btnPlay_clicked()
{
    WindowHolder::instance(new GameModesMenuWindow())->show();
    qDebug("game types showing");

}

void MenuWindow::on_btnSettings_clicked()
{
    WindowHolder::instance(new SettingsWindow())->show();
    qDebug("settings showing");
}

void MenuWindow::keyPressEvent(QKeyEvent *event) {
    if (event->key() == Qt::Key_Escape) {
        WindowHolder::holdPrevWidget();
```

```
        WindowHolder::instance(this)->close();

        qDebug() << "escape pressed from" << this;
    }
}
```

settingswindow.cpp

```
#include "settingswindow.h"
#include "ui_settingswindow.h"
#include "windowholder.h"

#include <string>
#include <QFileDialog>
#include <QErrorMessage>

#include "../utils.h"

SettingsWindow::SettingsWindow(QWidget *parent)
    : QWidget(parent)
    , ui(new Ui::SettingsWindow) {
    ui->setupUi(this);

    SettingsJson settingsJson;
    if (SettingsJson::updateParametresFromJsonFile(settingsJson))
        qDebug() << "update parametres success";
    else {
        qDebug() << "update not completed";
        (new QErrorMessage(this))->showMessage(QString("Получить настройки не
удалось!"));
    }

    switch (settingsJson.colorFigureInBottom){
        case FigureColor::WHITE:
            ui->rBtnDownLocateWhite->setChecked(true);
            break;

        case FigureColor::BLACK:
            ui->rBtnDownLocateBlack->setChecked(true);
            break;

        case FigureColor::DEFAULT:
            ui->rBtnDownLocateDefault->setChecked(true);
            break;
    }

    ui->lEditMinOnTimer->setText(QString::number(settingsJson.minOnTimer));
    ui->lEditSecOnTimer->setText(QString::number(settingsJson.secOnTimer));

    switch (settingsJson.figureDisposition){
    case FigureDisposition::CLASSIC:
        on_rBtnClassicDisposition_clicked();
        break;

    case FigureColor::BLACK:
        on_rBtnCustomDisposition_clicked();
        break;
    }

    ui->lEditFileNameToFigureDisposition-
>setText(Utils::toQString(settingsJson.fileNameOfFigureDisposition));
}
```

```
SettingsWindow::~SettingsWindow() {
    delete ui;
}

void SettingsWindow::keyPressEvent(QKeyEvent *event) {
    if (event->key() == Qt::Key_Escape) {
        WindowHolder::holdPrevWidget();

        qDebug() << "escape pressed from" << this;
    }
}

void SettingsWindow::on_btnSave_clicked() {
    SettingsJson settingsJson;
    if (ui->rBtnDownLocateWhite->isChecked())
        settingsJson.colorFigureInBottom = FigureColor::WHITE;
    else if (ui->rBtnDownLocateBlack->isChecked())
        settingsJson.colorFigureInBottom = FigureColor::BLACK;
    else if (ui->rBtnDownLocateDefault->isChecked())
        settingsJson.colorFigureInBottom = FigureColor::DEFAULT;

    int minutes = ui->lEditMinOnTimer->text().toInt();
    if (minutes < 0 || minutes > 150)
        ui->lEditMinOnTimer->setText(Utils::toCharArr(settingsJson.minOnTimer));
    else
        settingsJson.minOnTimer = minutes;

    int second = ui->lEditSecOnTimer->text().toInt();
    if (second < 0 || second > 59)
        ui->lEditSecOnTimer->setText(Utils::toCharArr(settingsJson.secOnTimer));
    else
        settingsJson.secOnTimer = second;

    if (ui->rBtnClassicDisposition->isChecked())
        settingsJson.figureDisposition = FigureDisposition::CLASSIC;
    else if (ui->rBtnCustomDisposition->isChecked())
        settingsJson.figureDisposition = FigureDisposition::CUSTOM;

    settingsJson.fileNameOfFigureDisposition = ui-
>lEditFileNameToFigureDisposition->text().toStdString();

    if (SettingsJson::saveParametresToJsonFile(settingsJson))
        qDebug() << "save correctly";
    else
        qDebug() << "not saved";
}


void SettingsWindow::on_btnSetFileNameToFigureDisposition_clicked()
{
    ui->lEditFileNameToFigureDisposition->setText(
        QFileDialog::getOpenFileName(
            this,
            QString(),
            QString(),
            "txt (*.txt)")
        );
}
```

```
void SettingsWindow::on_rBtnClassicDisposition_clicked()
{
    ui->rBtnClassicDisposition->setChecked(true);
    ui->lEditFileNameToFigureDisposition->setDisabled(true);
    ui->btnSetFileNameToFigureDisposition->setDisabled(true);
}


void SettingsWindow::on_rBtnCustomDisposition_clicked()
{
    ui->rBtnCustomDisposition->setChecked(true);
    ui->lEditFileNameToFigureDisposition->setEnabled(true);
    ui->btnSetFileNameToFigureDisposition->setEnabled(true);
}
```

main.cpp

```
#include "window/menuwindow.h"
#include "window/windowholder.h"

#include <QApplication>
#include <QtLogging>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    WindowHolder::instance(new MenuWindow())->show();
    qDebug("menu showing");

    return a.exec();
}
```