**Файл ActionButton.h**

#ifndef ACTIONBUTTON\_H

#define ACTIONBUTTON\_H

#include <QObject>

#include <QGraphicsRectItem>

#include <QGraphicsSceneMouseEvent>

#include <QGraphicsSceneHoverEvent>

class ActionButton: public QObject, public QGraphicsRectItem {

Q\_OBJECT

public:

ActionButton(QString title);

void mousePressEvent(QGraphicsSceneMouseEvent \*event);

void hoverEnterEvent(QGraphicsSceneHoverEvent \*event);

void hoverLeaveEvent(QGraphicsSceneHoverEvent \*event);

signals:

void buttonPressed();

private:

void setBackgroundColor(Qt::GlobalColor color);

};

#endif // ACTIONBUTTON\_H

**Файл basepawnmodel.h**

#ifndef PAWNMODEL\_H

#define PAWNMODEL\_H

#include <QString>

#include "boardposition.h"

enum class PawnType {

king,

queen,

rook,

bishop,

knight,

pawn

};

enum class PlayerType {

black,

white

};

class BasePawnModel {

public:

BasePawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath);

BoardPosition position;

PlayerType owner;

PawnType type;

QString imagePath;

bool didTakeFirstMove;

virtual bool validateMove(BoardPosition positionToMove,

BasePawnModel \*pawnOnPositionToMove,

BoardPosition \*requestedActivePawnPosition) = 0;

virtual ~BasePawnModel() = default;

protected:

bool pawnWantsToMoveByOneField(BoardPosition positionToMove);

bool validateDiagonalMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove);

bool validateVerticalOrHorizontalMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove);

};

#endif // PAWNMODEL\_H

**Файл bishoppawnmodel.h**

#ifndef BISHOPPAWNMODEL\_H

#define BISHOPPAWNMODEL\_H

#include "basepawnmodel.h"

class BishopPawnModel: public BasePawnModel {

public:

BishopPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath);

bool validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition);

};

#endif // BISHOPPAWNMODEL\_H

**Файл boardfield.h**

#ifndef BOARDFIELD\_H

#define BOARDFIELD\_H

#include <QGraphicsRectItem>

#include <QGraphicsSceneMouseEvent>

#include <boardposition.h>

class BoardField: public QGraphicsRectItem {

public:

BoardField(QColor backgroundColor,

BoardPosition position,

QGraphicsItem \*parent = nullptr);

static int defaultWidthHeight;

BoardPosition getPosition();

private:

BoardPosition position;

};

#endif // BOARDFIELD\_H

**Файл boardframefield.h**

#ifndef BOARDFRAMEFIELD\_H

#define BOARDFRAMEFIELD\_H

#include <QGraphicsRectItem>

class BoardFrameField: public QGraphicsRectItem {

public:

BoardFrameField(QGraphicsItem \*parent = nullptr);

void setTitle(QString title);

};

#endif // BOARDFRAMEFIELD\_H

**Файл boardposition.h**

#ifndef BOARDPOSITION\_H

#define BOARDPOSITION\_H

struct BoardPosition {

int x;

int y;

};

#endif // BOARDPOSITION\_H

**Файл boardview.h**

#ifndef BOARDVIEW\_H

#define BOARDVIEW\_H

#include <QGraphicsRectItem>

#include <QList>

#include <QPoint>

#include "boardfield.h"

#include "pawnfield.h"

class BoardView: public QGraphicsRectItem {

public:

BoardView();

static int numberOfRowsColumns;

static int startXPosition;

static int startYPosition;

QList<BoardField\*> getFields();

void draw();

void initializePawnFields(QList<BasePawnModel\*> pawns);

PawnField\* getPawnAtBoardPosition(BoardPosition boardPosition);

PawnField\* getPawnAtMousePosition(QPoint point);

void moveActivePawnToMousePosition(QPoint point, BasePawnModel \*pawn);

void placeActivePawnAtBoardPosition(BasePawnModel \*pawn, BoardPosition boardPosition);

void removePawnAtBoardPosition(BoardPosition boardPosition);

void setPawnMoveCheckWarning(bool visible);

void promotePawnAtBoardPosition(BoardPosition boardPosition);

private:

QList<BoardField\*> fields;

QList<PawnField\*> pawns;

QGraphicsTextItem \*checkWarningTitleTextItem;

QGraphicsTextItem \*checkWarningDescriptionTextItem;

void placeBoardFields();

void createFieldsColumn(int xPosition, int columnNumber);

void drawBoardFrame();

void drawBoardFrameAtPosition(QPoint point, QRectF rect, QString title);

void drawCheckWarningTextItems();

QPointF getCoordinatesForBoardPosition(BoardPosition position);

};

#endif // BOARDVIEW\_H

**Файл boardviewmodel.h**

#ifndef BOARDVIEWMODEL\_H

#define BOARDVIEWMODEL\_H

#include <QPoint>

#include "boardposition.h"

#include "pawnfield.h"

#include "basepawnmodel.h"

#include "pawnviewmodel.h"

class BoardViewModel {

public:

BoardViewModel();

bool isEnPassantAvailable;

QList<BasePawnModel\*> getBlackPawns();

QList<BasePawnModel\*> getWhitePawns();

BasePawnModel\* getActivePawn();

PlayerType getWhosTurn();

PlayerType\* getWinner();

void setActivePawnForField(PawnField \*pawn);

void setNewPositionForActivePawn(BoardPosition position);

void discardActivePawn();

BoardPosition getBoardPositionForMousePosition(QPoint position);

bool validatePawnPalcementForMousePosition(QPoint position);

bool validatePawnMove(BoardPosition positionToMove, BasePawnModel \*pawnToValidate = nullptr, BoardPosition \*requestedActivePawnPosition = nullptr);

bool didRemoveEnemyOnBoardPosition(BoardPosition boardPosition);

bool isKingInCheck(PlayerType owner, bool isCheckingActivePlayer, BoardPosition positionToMoveActivePlayer);

bool didPromoteActivePawn();

void switchRound();

private:

BasePawnModel \*activePawn;

PlayerType whosTurn;

QList<BasePawnModel\*> blackPawns;

QList<BasePawnModel\*> whitePawns;

PawnViewModel pawnViewModel;

PlayerType \*winner;

void initializePawns();

void initializePawnsForRow(int rowNumber, PlayerType owner);

BasePawnModel\* getPawnOnBoardPosition(BoardPosition baordPosition);

bool validateAnotherPawnIntersection(BoardPosition positionToMove, BasePawnModel \*pawnToValidate, BoardPosition \*requestedActivePawnPosition = nullptr);

bool validateKingsCheckForPawns(QList<BasePawnModel\*> pawns, bool isCheckingActivePlayer, BasePawnModel \*king, BoardPosition positionToMoveActivePlayer);

};

#endif // BOARDVIEWMODEL\_H

**Файл congratulationsview.h**

#ifndef CONGRATULATIONSVIEW\_H

#define CONGRATULATIONSVIEW\_H

#include <QGraphicsRectItem>

#include "basepawnmodel.h"

class CongratulationsView: public QObject, public QGraphicsRectItem {

Q\_OBJECT

public:

CongratulationsView(PlayerType winner);

};

#endif // CONGRATULATIONSVIEW\_H

**Файл constants.h**

#ifndef CONSTANTS\_H

#define CONSTANTS\_H

#include <QColor>

class Constants {

public:

static int defaultMargin;

static QColor defaultTextColor;

};

#endif // CONSTANTS\_H

**Файл gameview.h**

#ifndef GAME\_H

#define GAME\_H

#include <QObject>

#include <QGraphicsView>

#include <QGraphicsScene>

#include <QMouseEvent>

#include "boardview.h"

#include "boardviewmodel.h"

#include "pawnfield.h"

#include "playerview.h"

class GameView : public QGraphicsView {

Q\_OBJECT

public:

GameView();

QGraphicsScene \*scene;

void displayMainMenu();

public slots:

void startGame();

void quitGame();

void resetGame();

private:

BoardViewModel boardViewModel;

bool gameStarted;

BoardView \*board;

PlayerView \*blackPlayerView;

PlayerView \*whitePlayerView;

void drawBoard();

void drawSettingsPanel();

void drawUserPanel();

PlayerView\* drawViewForUser(PlayerType player);

void drawTitle(double yPosition, int fontSize);

void mousePressEvent(QMouseEvent \*event);

void mouseMoveEvent(QMouseEvent \*event);

void selectPawn(PawnField \*pawn);

void handleSelectingPointForActivePawnByMouse(QPoint point);

void setCheckStateOnPlayerView(PlayerType player, bool isInCheck);

void moveActivePawnToSelectedPoint(QPoint point);

void releaseActivePawn();

void showCongratulationsScreen(PlayerType winner);

};

#endif // GAME\_H

**Файл kingpawnmodel.h**

#ifndef KINGPAWNMODEL\_H

#define KINGPAWNMODEL\_H

#include "basepawnmodel.h"

class KingPawnModel: public BasePawnModel {

public:

KingPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath);

bool validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition);

};

#endif // KINGPAWNMODEL\_H

**Файл knightpawnmodel.h**

#ifndef KNIGHTPAWNMODEL\_H

#define KNIGHTPAWNMODEL\_H

#include "basepawnmodel.h"

class KnightPawnModel: public BasePawnModel {

public:

KnightPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath);

bool validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition);

};

#endif // KNIGHTPAWNMODEL\_H

**Файл pawnfield.h**

#ifndef PAWN\_H

#define PAWN\_H

#include <QGraphicsRectItem>

#include <QLabel>

#include "boardposition.h"

#include "basepawnmodel.h"

class PawnField: public QGraphicsRectItem {

public:

PawnField(BoardPosition position,

QString imagePath,

QGraphicsItem \*parent = nullptr);

BoardPosition getPosition();

void setPosition(BoardPosition position);

void setImage(QString imagePath);

private:

BoardPosition position;

QLabel \*imageLabel;

QPixmap image;

};

#endif // PAWN\_H

**Файл pawnpawnmodel.h**

#ifndef PAWNPAWNMODEL\_H

#define PAWNPAWNMODEL\_H

#include "basepawnmodel.h"

class PawnPawnModel: public BasePawnModel {

public:

PawnPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath);

bool validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition);

};

#endif // PAWNPAWNMODEL\_H

**Файл pawnviewmodel.h**

#ifndef PAWNMANAGER\_H

#define PAWNMANAGER\_H

#include <QString>

#include "basepawnmodel.h"

class PawnViewModel {

public:

PawnViewModel();

QString getImagePath(PawnType type, PlayerType owner);

PawnType getTypeForInitialPosition(BoardPosition position);

};

#endif // PAWNMANAGER\_H

**Файл playerview.h**

#ifndef PLAYERVIEW\_H

#define PLAYERVIEW\_H

#include <QGraphicsItem>

#include <QGraphicsRectItem>

#include "basepawnmodel.h"

class PlayerView: public QGraphicsRectItem {

public:

PlayerView(QGraphicsItem \*parent = nullptr);

static int defaultWidthHeight;

void setPlayer(PlayerType owner);

void setActive(bool active);

void setIsInCheck(bool isCheck);

private:

QGraphicsTextItem \*checkTextItem;

};

#endif // PLAYERVIEW\_H

**Файл queenpawnmodel.h**

#ifndef QUEENPAWNMODEL\_H

#define QUEENPAWNMODEL\_H

#include "basepawnmodel.h"

class QueenPawnModel: public BasePawnModel {

public:

QueenPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath);

bool validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition);

};

#endif // QUEENPAWNMODEL\_H

**Файл rookpawnmodel.h**

#ifndef ROOKPAWNMODEL\_H

#define ROOKPAWNMODEL\_H

#include "basepawnmodel.h"

class RookPawnModel: public BasePawnModel {

public:

RookPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath);

bool validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition);

};

#endif // ROOKPAWNMODEL\_H

**Файл utils.h**

#ifndef UTILS\_H

#define UTILS\_H

#include <QAbstractGraphicsShapeItem>

#include <QColor>

class Utils {

public:

static void setBackgroundColor(QColor color, QAbstractGraphicsShapeItem \*item);

static void setImage(QString imagePath, QGraphicsRectItem \*item);

static QGraphicsTextItem\* createTextItem(QString title, int fontSize, QColor textColor, QGraphicsItem \*parent = nullptr);

};

#endif // UTILS\_H

**Файл actionbutton.cpp**

#include "actionbutton.h"

#include <QBrush>

#include <QGraphicsRectItem>

#include <QFont>

#include "constants.h"

#include "utils.h"

ActionButton::ActionButton(QString title) {

setRect(0, 0, 200, 50);

QColor backgroundColor = QColor(157, 128, 101);

Utils::setBackgroundColor(backgroundColor, this);

QColor textColor = QColor(44, 41, 51);

QGraphicsTextItem \*text = Utils::createTextItem(title, 20, textColor, this);

double xPosition = rect().width()/2 - text->boundingRect().width()/2;

double yPosition = rect().height()/2 - text->boundingRect().height()/2;

text->setPos(xPosition, yPosition);

// allow responding to hover events

setAcceptHoverEvents(true);

}

void ActionButton::mousePressEvent(QGraphicsSceneMouseEvent \*event) {

emit buttonPressed();

}

void ActionButton::hoverEnterEvent(QGraphicsSceneHoverEvent \*event) {

QColor backgroundColor = QColor(196, 178, 140);

Utils::setBackgroundColor(backgroundColor, this);

}

void ActionButton::hoverLeaveEvent(QGraphicsSceneHoverEvent \*event) {

QColor backgroundColor = QColor(157, 128, 101);

Utils::setBackgroundColor(backgroundColor, this);

}

**Файл basepawnmodel.cpp**

#include "basepawnmodel.h"

BasePawnModel::BasePawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath) {

this->position = position;

this->owner = owner;

this->type = type;

this->imagePath = imagePath;

didTakeFirstMove = false;

}

bool BasePawnModel::pawnWantsToMoveByOneField(BoardPosition positionToMove) {

int xDiference = positionToMove.x - this->position.x;

int yDiference = positionToMove.y - this->position.y;

int numbeOfFieldsToMove = std::max(abs(xDiference), abs(yDiference));

return (numbeOfFieldsToMove == 1);

}

bool BasePawnModel::validateDiagonalMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove) {

if (pawnOnPositionToMove && pawnOnPositionToMove->owner == this->owner) {

return false;

}

int xDiference = positionToMove.x - this->position.x;

int yDiference = positionToMove.y - this->position.y;

if (abs(xDiference) != abs(yDiference)) {

return false;

}

return true;

}

bool BasePawnModel::validateVerticalOrHorizontalMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove) {

if (pawnOnPositionToMove && pawnOnPositionToMove->owner == this->owner) {

return false;

}

if ((positionToMove.x != this->position.x && positionToMove.y != this->position.y)) {

return false;

}

return true;

}

**Файл bishoppawnmodel.cpp**

#include "bishoppawnmodel.h"

BishopPawnModel::BishopPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath): BasePawnModel (position, owner, type, imagePath) {}

bool BishopPawnModel::validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) {

if (pawnOnPositionToMove && pawnOnPositionToMove->owner == this->owner) {

return false;

}

return validateDiagonalMove(positionToMove, pawnOnPositionToMove);

}

**Файл boardfield.cpp**

#include "boardfield.h"

#include <utils.h>

#include <gameview.h>

int BoardField::defaultWidthHeight = 60;

extern GameView \*game;

BoardField::BoardField(QColor backgroundColor,

BoardPosition position,

QGraphicsItem \*parent): QGraphicsRectItem(parent) {

this->position = position;

Utils::setBackgroundColor(backgroundColor, this);

setPen(Qt::NoPen);

setAcceptHoverEvents(true);

}

BoardPosition BoardField::getPosition() {

return position;

}

**Файл boardframefield.cpp**

#include "boardframefield.h"

#include "constants.h"

#include "utils.h"

#include "gameview.h"

#include <QFont>

extern GameView \*game;

BoardFrameField::BoardFrameField(QGraphicsItem \*parent): QGraphicsRectItem(parent) {

QColor backgroundColor = QColor(55, 51, 63);

Utils::setBackgroundColor(backgroundColor, this);

setPen(Qt::NoPen);

}

void BoardFrameField::setTitle(QString title) {

QGraphicsTextItem \*titleItem = Utils::createTextItem(title, 16, Constants::defaultTextColor);

double titleXPosition = this->pos().x() + this->boundingRect().width()/2 - titleItem->boundingRect().width()/2;

double titleYPosition = this->pos().y() + this->boundingRect().height()/2 - titleItem->boundingRect().height()/2;;

titleItem->setPos(titleXPosition, titleYPosition);

game->scene->addItem(titleItem);

}

**Файл boardview.cpp**

#include "boardview.h"

#include <QLabel>

#include "boardfield.h"

#include "boardposition.h"

#include "constants.h"

#include "boardframefield.h"

#include "gameview.h"

#include "pawnfield.h"

#include "utils.h"

extern GameView \*game;

int BoardView::numberOfRowsColumns = 8;

int BoardView::startXPosition = 100;

int BoardView::startYPosition = 150;

BoardView::BoardView() {

int size = numberOfRowsColumns \* BoardField::defaultWidthHeight;

setRect(startXPosition, startYPosition, size, size);

game->scene->addItem(this);

}

QList<BoardField\*> BoardView::getFields() {

return fields;

}

void BoardView::draw() {

placeBoardFields();

drawBoardFrame();

drawCheckWarningTextItems();

}

void BoardView::initializePawnFields(QList<BasePawnModel\*> pawns) {

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

BasePawnModel \*pawnModel = pawns[i];

PawnField \*pawn = new PawnField(pawnModel->position, pawnModel->imagePath, this);

int pawnXPosition = startXPosition + pawnModel->position.x \* BoardField::defaultWidthHeight;

int pawnYPosition = startYPosition + pawnModel->position.y \* BoardField::defaultWidthHeight;

pawn->setRect(0, 0, BoardField::defaultWidthHeight, BoardField::defaultWidthHeight);

pawn->setPos(pawnXPosition, pawnYPosition);

this->pawns.append(pawn);

}

}

void BoardView::moveActivePawnToMousePosition(QPoint point, BasePawnModel \*pawn) {

int xPosition = point.x() - BoardField::defaultWidthHeight/2;

int yPosition = point.y() - BoardField::defaultWidthHeight/2;

PawnField \*pawnField = getPawnAtBoardPosition(pawn->position);

if (pawnField) {

pawnField->setPos(xPosition, yPosition);

}

}

void BoardView::placeActivePawnAtBoardPosition(BasePawnModel \*pawn, BoardPosition boardPosition) {

PawnField \*pawnField = getPawnAtBoardPosition(pawn->position);

if (pawnField) {

QPointF coordinates = getCoordinatesForBoardPosition(boardPosition);

pawnField->setZValue(0);

pawnField->setPos(coordinates);

pawnField->setPosition(boardPosition);

}

}

void BoardView::removePawnAtBoardPosition(BoardPosition boardPosition) {

PawnField \*pawnField = getPawnAtBoardPosition(boardPosition);

game->scene->removeItem(pawnField);

int index = pawns.indexOf(pawnField);

pawns.removeAt(index);

delete pawnField;

}

void BoardView::setPawnMoveCheckWarning(bool visible) {

int opacity = visible ? 1 : 0;

checkWarningTitleTextItem->setOpacity(opacity);

checkWarningDescriptionTextItem->setOpacity(opacity);

}

void BoardView::promotePawnAtBoardPosition(BoardPosition boardPosition) {

PawnField \*pawn = getPawnAtBoardPosition(boardPosition);

QString imageFileName;

if (pawn->getPosition().y == 7) {

imageFileName = ":Images/queen\_black.svg";

} else {

imageFileName = ":Images/queen\_white.svg";

}

pawn->setImage(imageFileName);

}

PawnField\* BoardView::getPawnAtMousePosition(QPoint point) {

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

PawnField \*pawn = pawns[i];

QPointF pawnPos = pawn->pos();

if ((point.x() < (pawnPos.x() + pawn->rect().width())) &&

(point.x() > pawnPos.x()) &&

(point.y() < (pawnPos.y() + pawn->rect().height())) &&

(point.y() > pawnPos.y())) {

return pawn;

}

}

return nullptr;

}

PawnField\* BoardView::getPawnAtBoardPosition(BoardPosition boardPosition) {

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

BoardPosition pawnPosition = pawns[i]->getPosition();

if (pawnPosition.x == boardPosition.x && pawnPosition.y == boardPosition.y) {

return pawns[i];

}

}

return nullptr;

}

void BoardView::placeBoardFields() {

for (int i = 0; i< numberOfRowsColumns; i++ ) {

int xPosition = i \* BoardField::defaultWidthHeight;

createFieldsColumn(xPosition, i);

}

}

// creates a column of fields at the specified location with specified number of rows

void BoardView::createFieldsColumn(int xPosition, int columnNumber) {

for (int rowNumber = 0; rowNumber < numberOfRowsColumns; rowNumber++) {

QColor backgroundColor;

if (columnNumber % 2 == 0) {

if (rowNumber % 2 == 0) {

backgroundColor = QColor(196, 178, 140);

} else {

backgroundColor = QColor(157, 128, 101);

}

} else {

if (rowNumber % 2 == 0) {

backgroundColor = QColor(157, 128, 101);

} else {

backgroundColor = QColor(196, 178, 140);

}

}

BoardPosition position = { columnNumber, rowNumber };

BoardField \*field = new BoardField(backgroundColor, position, this);

int filedYPosition = startYPosition + rowNumber \* BoardField::defaultWidthHeight;

field->setRect(xPosition + startXPosition,

filedYPosition,

BoardField::defaultWidthHeight,

BoardField::defaultWidthHeight);

fields.append(field);

}

}

void BoardView::drawBoardFrame() {

QString lettersTitles[] = {"A", "B", "C", "D", "E", "F", "G", "H"};

QString numberTitles[] = {"1", "2", "3", "4", "5", "6", "7", "8"};

for (int i = 0; i< numberOfRowsColumns; i++ ) {

int xPosition = startXPosition + i \* BoardField::defaultWidthHeight;

QPoint point = QPoint(xPosition, startYPosition - 30);

QRectF rect = QRectF(0, 0, BoardField::defaultWidthHeight, 30);

drawBoardFrameAtPosition(point, rect, lettersTitles[i]);

}

for (int i = 0; i< numberOfRowsColumns; i++ ) {

int xPosition = startXPosition + i \* BoardField::defaultWidthHeight;

int yPosition = startYPosition + numberOfRowsColumns \* BoardField::defaultWidthHeight;

QPoint point = QPoint(xPosition, yPosition);

QRectF rect = QRectF(0, 0, BoardField::defaultWidthHeight, 30);

drawBoardFrameAtPosition(point, rect, lettersTitles[i]);

}

for (int i = 0; i< numberOfRowsColumns; i++ ) {

int yPosition = startYPosition + i \* BoardField::defaultWidthHeight;

QPoint point = QPoint(70, yPosition);

QRectF rect = QRectF(0, 0, 30, BoardField::defaultWidthHeight);

drawBoardFrameAtPosition(point, rect, numberTitles[i]);

}

for (int i = 0; i< numberOfRowsColumns; i++ ) {

int xPosition = startXPosition + numberOfRowsColumns \* BoardField::defaultWidthHeight;

int yPosition = startYPosition + i \* BoardField::defaultWidthHeight;

QPoint point = QPoint(xPosition, yPosition);

QRectF rect = QRectF(0, 0, 30, BoardField::defaultWidthHeight);

drawBoardFrameAtPosition(point, rect, numberTitles[i]);

}

}

void BoardView::drawBoardFrameAtPosition(QPoint point, QRectF rect, QString title) {

BoardFrameField \*frameField = new BoardFrameField(this);

frameField->setRect(rect);

frameField->setPos(point);

frameField->setTitle(title);

}

void BoardView::drawCheckWarningTextItems() {

checkWarningTitleTextItem = Utils::createTextItem("This move is not possible!", 18, Constants::defaultTextColor, this);

double titleXPosition = startXPosition + (BoardField::defaultWidthHeight\*numberOfRowsColumns)/2 - checkWarningTitleTextItem->boundingRect().width()/2;

double titleYPosition = startYPosition + (BoardField::defaultWidthHeight\*numberOfRowsColumns) + 40;

checkWarningTitleTextItem->setPos(titleXPosition, titleYPosition);

checkWarningTitleTextItem->setOpacity(0);

checkWarningDescriptionTextItem = Utils::createTextItem("You cannot make any move that places your own king in check", 18, Constants::defaultTextColor, this);

double descriptionXPosition = startXPosition + (BoardField::defaultWidthHeight\*numberOfRowsColumns)/2 - checkWarningDescriptionTextItem->boundingRect().width()/2;

double descriptionYPosition = startYPosition + (BoardField::defaultWidthHeight\*numberOfRowsColumns) + 60;

checkWarningDescriptionTextItem->setPos(descriptionXPosition, descriptionYPosition);

checkWarningDescriptionTextItem->setOpacity(0);

}

QPointF BoardView::getCoordinatesForBoardPosition(BoardPosition position) {

int xPosition = startXPosition + position.x\*BoardField::defaultWidthHeight;

int yPosition = startYPosition + position.y\*BoardField::defaultWidthHeight;

return QPointF(xPosition, yPosition);

}

**Файл boardviewmodel.cpp**

#include "boardviewmodel.h"

#include "boardview.h"

#include "boardfield.h"

#include "kingpawnmodel.h"

#include "queenpawnmodel.h"

#include "rookpawnmodel.h"

#include "bishoppawnmodel.h"

#include "knightpawnmodel.h"

#include "pawnpawnmodel.h"

#include <math.h>

BoardViewModel::BoardViewModel() {

activePawn = nullptr;

whosTurn = PlayerType::black;

isEnPassantAvailable = false;

pawnViewModel = PawnViewModel();

winner = nullptr;

initializePawns();

}

QList<BasePawnModel\*> BoardViewModel::getBlackPawns() {

return blackPawns;

}

QList<BasePawnModel\*> BoardViewModel::getWhitePawns() {

return whitePawns;

}

BasePawnModel\* BoardViewModel::getActivePawn() {

return activePawn;

}

PlayerType BoardViewModel::getWhosTurn() {

return whosTurn;

}

PlayerType\* BoardViewModel::getWinner() {

return winner;

}

void BoardViewModel::setActivePawnForField(PawnField \*pawn) {

BasePawnModel\* pawnModel = getPawnOnBoardPosition(pawn->getPosition());

if (pawnModel && pawnModel->owner == whosTurn) {

activePawn = pawnModel;

pawn->setZValue(1);

}

}

void BoardViewModel::setNewPositionForActivePawn(BoardPosition position) {

activePawn->didTakeFirstMove = true;

activePawn->position = position;

}

void BoardViewModel::discardActivePawn() {

activePawn = nullptr;

}

BasePawnModel\* BoardViewModel::getPawnOnBoardPosition(BoardPosition baordPosition) {

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

BasePawnModel \*pawnModel = blackPawns[i];

if (baordPosition.x == pawnModel->position.x &&

baordPosition.y == pawnModel->position.y) {

return pawnModel;

}

}

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

BasePawnModel \*pawnModel = whitePawns[i];

if (baordPosition.x == pawnModel->position.x &&

baordPosition.y == pawnModel->position.y) {

return pawnModel;

}

}

return nullptr;

}

bool BoardViewModel::validatePawnPalcementForMousePosition(QPoint point) {

if (point.x() > BoardView::startXPosition &&

point.x() < (BoardView::startXPosition + BoardField::defaultWidthHeight\*BoardView::numberOfRowsColumns) &&

point.y() > BoardView::startYPosition &&

point.y() < (BoardView::startYPosition + BoardField::defaultWidthHeight\*BoardView::numberOfRowsColumns)) {

return true;

}

return false;

}

bool BoardViewModel::validatePawnMove(BoardPosition positionToMove,

BasePawnModel \*pawn,

BoardPosition \*requestedActivePawnPosition) {

BasePawnModel \*pawnToValidate;

if (pawn) {

pawnToValidate = pawn;

} else {

pawnToValidate = activePawn;

}

BasePawnModel \*pawnOnPositionToMove = getPawnOnBoardPosition(positionToMove);

bool isMoveValid = pawnToValidate->validateMove(positionToMove, pawnOnPositionToMove, requestedActivePawnPosition);

switch (pawnToValidate->type) {

case PawnType::king:

case PawnType::queen:

case PawnType::rook:

case PawnType::bishop:

case PawnType::pawn:

return isMoveValid && validateAnotherPawnIntersection(positionToMove, pawnToValidate, requestedActivePawnPosition);

case PawnType::knight:

return isMoveValid;

}

}

bool BoardViewModel::didRemoveEnemyOnBoardPosition(BoardPosition boardPosition) {

BasePawnModel \*pawn = getPawnOnBoardPosition(boardPosition);

if (pawn && pawn->owner == whosTurn) {

return false;

}

if (pawn) {

switch (whosTurn) {

case PlayerType::black: {

int index = whitePawns.indexOf(pawn);

whitePawns.removeAt(index);

}

break;

case PlayerType::white: {

int index = blackPawns.indexOf(pawn);

blackPawns.removeAt(index);

}

break;

}

if (pawn->type == PawnType::king) {

winner = &whosTurn;

}

delete pawn;

return true;

}

return false;

}

bool BoardViewModel::isKingInCheck(PlayerType owner,

bool isCheckingActivePlayer,

BoardPosition positionToMoveActivePlayer) {

BasePawnModel \*king = nullptr;

if (isCheckingActivePlayer && activePawn->type == PawnType::king) {

king = activePawn;

} else {

switch (owner) {

case PlayerType::black:

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

BasePawnModel \*pawn = blackPawns[i];

if (pawn->type == PawnType::king) {

king = pawn;

}

}

break;

case PlayerType::white:

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

BasePawnModel \*pawn = whitePawns[i];

if (pawn->type == PawnType::king) {

king = pawn;

}

}

break;

}

}

if (king) {

bool isInCheck = false;

switch (owner) {

case PlayerType::black:

isInCheck = validateKingsCheckForPawns(whitePawns, isCheckingActivePlayer, king, positionToMoveActivePlayer);

break;

case PlayerType::white:

isInCheck = validateKingsCheckForPawns(blackPawns, isCheckingActivePlayer, king, positionToMoveActivePlayer);

}

return isInCheck;

}

return false;

}

bool BoardViewModel::didPromoteActivePawn() {

if (!activePawn) {

return false;

}

if (activePawn->type != PawnType::pawn) {

return false;

}

switch (activePawn->owner) {

case PlayerType::black:

if (activePawn->position.y == 7) {

activePawn->type = PawnType::queen;

return true;

}

break;

case PlayerType::white:

if (activePawn->position.y == 0) {

activePawn->type = PawnType::queen;

return true;

}

break;

}

return false;

}

void BoardViewModel::switchRound() {

switch (whosTurn) {

case PlayerType::black:

whosTurn = PlayerType::white;

break;

case PlayerType::white:

whosTurn = PlayerType::black;

break;

}

}

BoardPosition BoardViewModel::getBoardPositionForMousePosition(QPoint point) {

int xPosition = static\_cast<int>(floor((point.x() - BoardView::startXPosition)/BoardField::defaultWidthHeight));

int yPosition = static\_cast<int>(floor((point.y() - BoardView::startYPosition)/BoardField::defaultWidthHeight));

return BoardPosition { xPosition, yPosition };

}

void BoardViewModel::initializePawns() {

initializePawnsForRow(0, PlayerType::black);

initializePawnsForRow(1, PlayerType::black);

initializePawnsForRow(6, PlayerType::white);

initializePawnsForRow(7, PlayerType::white);

}

void BoardViewModel::initializePawnsForRow(int rowNumber, PlayerType owner) {

for (int i = 0; i < BoardView::numberOfRowsColumns; i++) {

BoardPosition boardPosition = { i, rowNumber };

PawnType type = pawnViewModel.getTypeForInitialPosition(boardPosition);

QString imagePath = pawnViewModel.getImagePath(type, owner);

BasePawnModel \*pawn;

switch (type) {

case PawnType::king:

pawn = new KingPawnModel(boardPosition, owner, type, imagePath);

break;

case PawnType::queen:

pawn = new QueenPawnModel(boardPosition, owner, type, imagePath);

break;

case PawnType::rook:

pawn = new RookPawnModel(boardPosition, owner, type, imagePath);

break;

case PawnType::bishop:

pawn = new BishopPawnModel(boardPosition, owner, type, imagePath);

break;

case PawnType::knight:

pawn = new KnightPawnModel(boardPosition, owner, type, imagePath);

break;

case PawnType::pawn:

pawn = new PawnPawnModel(boardPosition, owner, type, imagePath);

break;

}

switch (owner) {

case PlayerType::black:

blackPawns.append(pawn);

break;

case PlayerType::white:

whitePawns.append(pawn);

break;

}

}

}

bool BoardViewModel::validateAnotherPawnIntersection(BoardPosition positionToMove,

BasePawnModel \*pawnToValidate,

BoardPosition \*requestedActivePawnPosition) {

int xDiference = positionToMove.x - pawnToValidate->position.x;

int yDiference = positionToMove.y - pawnToValidate->position.y;

int numbeOfFieldsToCheck = std::max(abs(xDiference), abs(yDiference));

if (numbeOfFieldsToCheck == 1) {

return true;

}

for (int i = 0; i < numbeOfFieldsToCheck; i++) {

BoardPosition positionToCheck;

if (xDiference < 0) {

if (yDiference == 0) {

positionToCheck = { pawnToValidate->position.x + (xDiference + i), pawnToValidate->position.y };

} else if (yDiference < 0) {

positionToCheck = { pawnToValidate->position.x + (xDiference + i), pawnToValidate->position.y + (yDiference + i) };

} else {

positionToCheck = { pawnToValidate->position.x + (xDiference + i), pawnToValidate->position.y + (yDiference - i) };

}

} else if (yDiference < 0) {

if (xDiference == 0) {

positionToCheck = { pawnToValidate->position.x, pawnToValidate->position.y + (yDiference + i) };

} else {

positionToCheck = { pawnToValidate->position.x + (xDiference - i), pawnToValidate->position.y + (yDiference + i) };

}

} else {

if (xDiference == 0) {

positionToCheck = { pawnToValidate->position.x, pawnToValidate->position.y + (yDiference - i) };

} else if (yDiference == 0) {

positionToCheck = { pawnToValidate->position.x + (xDiference - i), pawnToValidate->position.y };

} else {

positionToCheck = { pawnToValidate->position.x + (xDiference - i), pawnToValidate->position.y + (yDiference - i) };

}

}

BasePawnModel \*pawnToCheck = getPawnOnBoardPosition(positionToCheck);

if (requestedActivePawnPosition &&

positionToCheck.x != positionToMove.x &&

positionToCheck.y != positionToMove.y &&

positionToCheck.x == requestedActivePawnPosition->x &&

positionToCheck.y == requestedActivePawnPosition->y) {

return false;

}

if (pawnToCheck &&

(pawnToCheck->position.x != positionToMove.x || pawnToCheck->position.y != positionToMove.y)) {

return false;

}

}

return true;

}

bool BoardViewModel::validateKingsCheckForPawns(QList<BasePawnModel\*> pawns,

bool isCheckingActivePlayer,

BasePawnModel \*king,

BoardPosition positionToMoveActivePlayer) {

bool isInCheck = false;

// check every oppisite players pawn for kings check

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

BasePawnModel \*pawn = pawns[i];

if (isCheckingActivePlayer && activePawn->type == PawnType::king) {

if (validatePawnMove(positionToMoveActivePlayer, pawn, &positionToMoveActivePlayer)) {

isInCheck = true;

}

} else if (isCheckingActivePlayer) {

if ((positionToMoveActivePlayer.x != pawn->position.x || positionToMoveActivePlayer.y != pawn->position.y) &&

validatePawnMove(king->position, pawn, &positionToMoveActivePlayer)) {

isInCheck = true;

}

} else if (validatePawnMove(king->position, pawn)) {

isInCheck = true;

}

}

return isInCheck;

}

**Файл congratulationsview.cpp**

#include "congratulationsview.h"

#include <QLabel>

#include <QGraphicsProxyWidget>

#include "actionbutton.h"

#include "constants.h"

#include "gameview.h"

#include "utils.h"

extern GameView \*game;

CongratulationsView::CongratulationsView(PlayerType winner) {

// set title

QGraphicsTextItem \*titleItem = Utils::createTextItem("Congratulations!", 50, Constants::defaultTextColor, this);

double titleXPosition = 600 - titleItem->boundingRect().width()/2;

double titleYPosition = 100;

titleItem->setPos(titleXPosition, titleYPosition);

// set image

QString imagePath = ":Images/confetti.svg";

QPixmap image(imagePath);

QLabel \*imageLabel = new QLabel();

QGraphicsProxyWidget \*pMyProxy = new QGraphicsProxyWidget(this);

imageLabel->setPixmap(image.scaled(200, 200, Qt::KeepAspectRatio));

imageLabel->setAttribute(Qt::WA\_TranslucentBackground);

pMyProxy->setWidget(imageLabel);

pMyProxy->setPos(500, 180);

// set winner label

QString winnerName = winner == PlayerType::black ? "Player black" : "Player white";

QGraphicsTextItem \*descriptionItem = Utils::createTextItem(winnerName + " has won!", 25, Constants::defaultTextColor, this);

double descriptionXPosition = 600 - descriptionItem->boundingRect().width()/2;

double descriptionYPosition = 400;

descriptionItem->setPos(descriptionXPosition, descriptionYPosition);

// add action button

ActionButton \*actionButton = new ActionButton("Quit game");

double buttonXPosition = 600 - actionButton->boundingRect().width()/2;

double buttonYPosition = 500;

actionButton->setPos(buttonXPosition, buttonYPosition);

connect(actionButton, SIGNAL(buttonPressed()), game, SLOT(quitGame()));

game->scene->addItem(actionButton);

game->scene->addItem(this);

}

**Файл constants.cpp**

#include "constants.h"

int Constants::defaultMargin = 30;

QColor Constants::defaultTextColor = QColor(157, 128, 101);

**Файл gameview.cpp**

#include "gameview.h"

#include <QGraphicsTextItem>

#include <QColor>

#include <QBrush>

#include "actionbutton.h"

#include "congratulationsview.h"

#include "constants.h"

#include "utils.h"

int viewWidth = 1200;

int viewHeight= 768;

GameView::GameView() {

setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);

setFixedSize(viewWidth, viewHeight);

scene = new QGraphicsScene();

scene->setSceneRect(0, 0, viewWidth, viewHeight);

setScene(scene);

QBrush brush;

brush.setStyle((Qt::SolidPattern));

QColor color = QColor(44, 41, 51);

brush.setColor(color);

scene->setBackgroundBrush(brush);

gameStarted = false;

}

void GameView::displayMainMenu() {

// create title label

double titleYPosition = 150;

drawTitle(titleYPosition, 50);

// create start button

ActionButton \*startButton = new ActionButton("Play");

double buttonXPosition = this->width()/2 - startButton->boundingRect().width()/2;

double buttonYPosition = 275;

startButton->setPos(buttonXPosition, buttonYPosition);

connect(startButton, SIGNAL(buttonPressed()), this, SLOT(startGame()));

scene->addItem(startButton);

// create quit button

ActionButton \*quitButton = new ActionButton("Quit");

double quitXPosition = this->width()/2 - quitButton->boundingRect().width()/2;

double quitYPosition = 350;

quitButton->setPos(quitXPosition, quitYPosition);

connect(quitButton, SIGNAL(buttonPressed()), this, SLOT(quitGame()));

scene->addItem(quitButton);

}

void GameView::startGame() {

scene->clear();

boardViewModel = BoardViewModel();

drawBoard();

drawSettingsPanel();

drawUserPanel();

int titleYPosition = Constants::defaultMargin;

drawTitle(titleYPosition, 40);

gameStarted = true;

}

void GameView::quitGame() {

close();

}

void GameView::resetGame() {

gameStarted = false;

scene->clear();

startGame();

}

void GameView::drawBoard() {

board = new BoardView();

board->draw();

board->initializePawnFields(boardViewModel.getBlackPawns());

board->initializePawnFields(boardViewModel.getWhitePawns());

}

void GameView::drawSettingsPanel() {

// create quit button

ActionButton \*resetButton = new ActionButton("Reset game");

double resetXPosition = 690 + resetButton->boundingRect().width()/2;

double resetYPosition = 420;

resetButton->setPos(resetXPosition, resetYPosition);

connect(resetButton, SIGNAL(buttonPressed()), this, SLOT(resetGame()));

scene->addItem(resetButton);

// create quit button

ActionButton \*quitButton = new ActionButton("Quit game");

double quitXPosition = 690 + quitButton->boundingRect().width()/2;

double quitYPosition = 490;

quitButton->setPos(quitXPosition, quitYPosition);

connect(quitButton, SIGNAL(buttonPressed()), this, SLOT(quitGame()));

scene->addItem(quitButton);

}

void GameView::drawUserPanel() {

blackPlayerView = drawViewForUser(PlayerType::black);

whitePlayerView = drawViewForUser(PlayerType::white);

blackPlayerView->setActive(true);

}

PlayerView\* GameView::drawViewForUser(PlayerType player) {

PlayerView \*playerView = new PlayerView();

int xPosition = 80;

int yPosition = BoardView::startYPosition;

switch (player) {

case PlayerType::black:

xPosition = 680;

break;

case PlayerType::white:

xPosition = 680 + PlayerView::defaultWidthHeight + 20;

break;

}

scene->addItem(playerView);

playerView->setRect(xPosition, yPosition, PlayerView::defaultWidthHeight, PlayerView::defaultWidthHeight);

playerView->setPlayer(player);

return playerView;

}

void GameView::drawTitle(double yPosition, int fontSize) {

QGraphicsTextItem \*title = Utils::createTextItem("Chess Game", fontSize, Qt::white);

double xPosition = this->width()/2 - title->boundingRect().width()/2;

title->setPos(xPosition, yPosition);

scene->addItem(title);

}

void GameView::mousePressEvent(QMouseEvent \*event) {

if (!gameStarted) {

QGraphicsView::mousePressEvent(event);

return;

} else if (event->button() == Qt::RightButton) {

releaseActivePawn();

} else if (boardViewModel.getActivePawn()) {

handleSelectingPointForActivePawnByMouse(event->pos());

} else {

PawnField \*pawn = board->getPawnAtMousePosition(event->pos());

selectPawn(pawn);

}

QGraphicsView::mousePressEvent(event);

}

void GameView::mouseMoveEvent(QMouseEvent \*event) {

// if there is a pawn selected, then make it follow the mouse

if (gameStarted && boardViewModel.getActivePawn()) {

board->moveActivePawnToMousePosition(event->pos(), boardViewModel.getActivePawn());

}

QGraphicsView::mouseMoveEvent(event);

}

void GameView::selectPawn(PawnField \*pawn) {

if (pawn == nullptr) {

return;

}

boardViewModel.setActivePawnForField(pawn);

}

void GameView::handleSelectingPointForActivePawnByMouse(QPoint point) {

if (boardViewModel.getActivePawn() == nullptr) {

return;

}

// check if mouse selected place on board

if (!boardViewModel.validatePawnPalcementForMousePosition(point)) {

return;

}

BoardPosition boardPosition = boardViewModel.getBoardPositionForMousePosition(point);

// first validate Move

if (!boardViewModel.validatePawnMove(boardPosition)) {

return;

}

// Players cannot make any move that places their own king in check

bool isKingInCheck = boardViewModel.isKingInCheck(boardViewModel.getActivePawn()->owner, true, boardPosition);

board->setPawnMoveCheckWarning(isKingInCheck);

if (isKingInCheck) {

return;

}

// check if field was taken by opposite player and remove it from the board

if (boardViewModel.didRemoveEnemyOnBoardPosition(boardPosition)) {

board->removePawnAtBoardPosition(boardPosition);

}

// move active pawn to new position

moveActivePawnToSelectedPoint(point);

// check if pawn can be promoted

if (boardViewModel.didPromoteActivePawn()) {

board->promotePawnAtBoardPosition(boardPosition);

}

// check for opposite player king's check

switch (boardViewModel.getActivePawn()->owner) {

case PlayerType::black:

setCheckStateOnPlayerView(PlayerType::white, boardViewModel.isKingInCheck(PlayerType::white, false, boardPosition));

break;

case PlayerType::white:

setCheckStateOnPlayerView(PlayerType::black, boardViewModel.isKingInCheck(PlayerType::black, false, boardPosition));

break;

}

// update active player check state

setCheckStateOnPlayerView(boardViewModel.getActivePawn()->owner, isKingInCheck);

// check if game is over

if (boardViewModel.getWinner()) {

showCongratulationsScreen(\*boardViewModel.getWinner());

return;

}

// change round owner to opposite player

boardViewModel.discardActivePawn();

boardViewModel.switchRound();

blackPlayerView->setActive(boardViewModel.getWhosTurn() == PlayerType::black);

whitePlayerView->setActive(boardViewModel.getWhosTurn() == PlayerType::white);

}

void GameView::setCheckStateOnPlayerView(PlayerType player, bool isInCheck) {

switch (player) {

case PlayerType::black:

blackPlayerView->setIsInCheck(isInCheck);

break;

case PlayerType::white:

whitePlayerView->setIsInCheck(isInCheck);

break;

}

}

// update pawn field position and pawn model position

void GameView::moveActivePawnToSelectedPoint(QPoint point) {

BoardPosition boardPosition = boardViewModel.getBoardPositionForMousePosition(point);

board->placeActivePawnAtBoardPosition(boardViewModel.getActivePawn(), boardPosition);

boardViewModel.setNewPositionForActivePawn(boardPosition);

}

void GameView::releaseActivePawn() {

if (boardViewModel.getActivePawn() == nullptr) {

return;

}

BasePawnModel \*activePawn = boardViewModel.getActivePawn();

board->placeActivePawnAtBoardPosition(activePawn, activePawn->position);

board->setPawnMoveCheckWarning(false);

boardViewModel.discardActivePawn();

}

void GameView::showCongratulationsScreen(PlayerType winner) {

gameStarted = false;

scene->clear();

CongratulationsView \*congratulationsView = new CongratulationsView(winner);

congratulationsView->setRect(0, 0, viewWidth, viewHeight);

}

**Файл kingpawnmodel.cpp**

#include "kingpawnmodel.h"

KingPawnModel::KingPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath): BasePawnModel(position, owner, type, imagePath) {}

bool KingPawnModel::validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) {

if (pawnOnPositionToMove && pawnOnPositionToMove->owner == this->owner) {

return false;

}

return pawnWantsToMoveByOneField(positionToMove);

}

**Файл knightpawnmodel.cpp**

#include "knightpawnmodel.h"

KnightPawnModel::KnightPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath): BasePawnModel (position, owner, type, imagePath) {}

bool KnightPawnModel::validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) {

if (pawnOnPositionToMove && pawnOnPositionToMove->owner == this->owner) {

return false;

}

int xDiference = abs(positionToMove.x - this->position.x);

int yDiference = abs(positionToMove.y - this->position.y);

if (xDiference == 2 && yDiference == 1) {

return true;

}

if (xDiference == 1 && yDiference == 2) {

return true;

}

return false;

}

**Файл main.cpp**

#include <QApplication>

#include <gameview.h>

GameView \*game;

int main(int argc, char \*argv[]) {

QApplication a(argc, argv);

game = new GameView();

game->show();

game->displayMainMenu();

return a.exec();

}

**Файл pawnfield.cpp**

#include "pawnfield.h"

#include <QGraphicsProxyWidget>

#include "boardfield.h"

#include "boardposition.h"

#include "gameview.h"

#include "utils.h"

extern GameView \*game;

PawnField::PawnField(BoardPosition position,

QString imagePath,

QGraphicsItem \*parent): QGraphicsRectItem(parent) {

this->position = position;

imageLabel = new QLabel();

image = QPixmap(imagePath);

QGraphicsProxyWidget \*pMyProxy = new QGraphicsProxyWidget(this);

imageLabel->setPixmap(image);

imageLabel->setAttribute(Qt::WA\_TranslucentBackground);

pMyProxy->setWidget(imageLabel);

setPen(Qt::NoPen);

}

void PawnField::setPosition(BoardPosition position) {

this->position = position;

}

void PawnField::setImage(QString imagePath) {

image.load(imagePath);

imageLabel->clear();

imageLabel->setPixmap(image);

}

BoardPosition PawnField::getPosition() {

return position;

}

**Файл pawnpawnmodel.cpp**

#include "pawnpawnmodel.h"

PawnPawnModel::PawnPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath): BasePawnModel (position, owner, type, imagePath) {}

bool PawnPawnModel::validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) {

if (pawnOnPositionToMove && pawnOnPositionToMove->owner == this->owner) {

return false;

}

int xDiference = positionToMove.x - this->position.x;

int yDiference = positionToMove.y - this->position.y;

int numbeOfFieldsToMove = std::max(abs(xDiference), abs(yDiference));

bool wantsToMoveByOneField = (numbeOfFieldsToMove == 1);

if (abs(xDiference) > 1 || abs(yDiference) > 2) {

return false;

}

if ( !wantsToMoveByOneField && this->didTakeFirstMove) {

return false;

}

bool wantsToMoveInGoodDirection;

switch (this->owner) {

case PlayerType::black:

wantsToMoveInGoodDirection = yDiference > 0;

break;

case PlayerType::white:

wantsToMoveInGoodDirection = yDiference < 0;

break;

}

if (wantsToMoveByOneField) {

if (requestedActivePawnPosition && xDiference == 0) {

return (wantsToMoveInGoodDirection &&

requestedActivePawnPosition->x != positionToMove.x &&

requestedActivePawnPosition->y != positionToMove.y);

} else if (xDiference == 0) {

return (wantsToMoveInGoodDirection && !pawnOnPositionToMove);

} else if (requestedActivePawnPosition) {

return (wantsToMoveInGoodDirection &&

requestedActivePawnPosition->x == positionToMove.x &&

requestedActivePawnPosition->y == positionToMove.y) || (wantsToMoveInGoodDirection && pawnOnPositionToMove);

} else {

return (wantsToMoveInGoodDirection && pawnOnPositionToMove);

}

}

return (wantsToMoveInGoodDirection &&

!this->didTakeFirstMove &&

xDiference == 0);

}

**Файл pawnviewmodel.cpp**

#include "pawnviewmodel.h"

PawnViewModel::PawnViewModel() {}

QString PawnViewModel::getImagePath(PawnType type, PlayerType owner) {

QString imageFileName;

switch (type) {

case PawnType::king:

if (owner == PlayerType::black) {

imageFileName = "king\_black.svg";

} else {

imageFileName = "king\_white.svg";

}

break;

case PawnType::queen:

if (owner == PlayerType::black) {

imageFileName = "queen\_black.svg";

} else {

imageFileName = "queen\_white.svg";

}

break;

case PawnType::rook:

if (owner == PlayerType::black) {

imageFileName = "rook\_black.svg";

} else {

imageFileName = "rook\_white.svg";

}

break;

case PawnType::bishop:

if (owner == PlayerType::black) {

imageFileName = "bishop\_black.svg";

} else {

imageFileName = "bishop\_white.svg";

}

break;

case PawnType::knight:

if (owner == PlayerType::black) {

imageFileName = "knight\_black.svg";

} else {

imageFileName = "knight\_white.svg";

}

break;

case PawnType::pawn:

if (owner == PlayerType::black) {

imageFileName = "pawn\_black.svg";

} else {

imageFileName = "pawn\_white.svg";

}

break;

}

return ":Images/" + imageFileName;

}

PawnType PawnViewModel::getTypeForInitialPosition(BoardPosition position) {

if (position.y == 1 || position.y == 6) {

return PawnType::pawn;

}

switch (position.x) {

case 0:

case 7:

return PawnType::rook;

case 1:

case 6:

return PawnType::knight;

case 2:

case 5:

return PawnType::bishop;

case 3:

return PawnType::queen;

case 4:

return PawnType::king;

}

return PawnType::pawn;

}

**Файл playerview.cpp**

#include "playerview.h"

#include <QObject>

#include <QFont>

#include "constants.h"

#include "gameview.h"

#include "utils.h"

int PlayerView::defaultWidthHeight = 200;

extern GameView \*game;

PlayerView::PlayerView(QGraphicsItem \*parent): QGraphicsRectItem(parent) {

QColor backgroundColor = QColor(55, 51, 63);

Utils::setBackgroundColor(backgroundColor, this);

setPen(Qt::NoPen);

}

void PlayerView::setPlayer(PlayerType owner) {

QString title;

QString imagePath;

switch (owner) {

case PlayerType::black:

title = "Black Player";

imagePath = ":Images/pawn\_black.svg";

break;

case PlayerType::white:

title = "White Player";

imagePath = ":Images/pawn\_white.svg";

break;

}

// set title

QGraphicsTextItem \*titleItem = Utils::createTextItem(title, 18, Constants::defaultTextColor, this);

double titleXPosition = this->boundingRect().x() + this->boundingRect().width()/2 - titleItem->boundingRect().width()/2;

double titleYPosition = this->boundingRect().y() + defaultWidthHeight - titleItem->boundingRect().height()/2 - Constants::defaultMargin;

titleItem->setPos(titleXPosition, titleYPosition);

// set image

PawnField \*pawn = new PawnField({ 0, 0 }, imagePath, this);

double pawnXPosition = this->boundingRect().x() + this->boundingRect().width()/2 - BoardField::defaultWidthHeight/2;

double pawnYPosition = this->boundingRect().y() + Constants::defaultMargin;

pawn->setRect(0, 0, BoardField::defaultWidthHeight, BoardField::defaultWidthHeight);

pawn->setPos(pawnXPosition, pawnYPosition);

// set check text item

checkTextItem = Utils::createTextItem("CHECK", 18, Constants::defaultTextColor, this);

double checkXPosition = this->boundingRect().x() + this->boundingRect().width()/2 - checkTextItem->boundingRect().width()/2;

double checkYPosition = this->boundingRect().y() + defaultWidthHeight - checkTextItem->boundingRect().height()/2 - Constants::defaultMargin\*2;

checkTextItem->setPos(checkXPosition, checkYPosition);

checkTextItem->setOpacity(0);

}

void PlayerView::setActive(bool active) {

QColor borderColor;

if (active) {

borderColor = QColor(157, 128, 101);

} else {

borderColor = QColor(55, 51, 63);

}

QPen pen(borderColor);

setPen(pen);

}

void PlayerView::setIsInCheck(bool isCheck) {

int opacity = isCheck ? 1 : 0;

checkTextItem->setOpacity(opacity);

}

**Файл queenpawnmodel.cpp**

#include "queenpawnmodel.h"

QueenPawnModel::QueenPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath): BasePawnModel (position, owner, type, imagePath) {}

bool QueenPawnModel::validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) {

if (pawnOnPositionToMove && pawnOnPositionToMove->owner == this->owner) {

return false;

}

if (validateDiagonalMove(positionToMove, pawnOnPositionToMove)) {

return true;

}

return validateVerticalOrHorizontalMove(positionToMove, pawnOnPositionToMove);

}

**Файл rookpawnmodel.cpp**

#include "rookpawnmodel.h"

RookPawnModel::RookPawnModel(BoardPosition position, PlayerType owner, PawnType type, QString imagePath): BasePawnModel (position, owner, type, imagePath) {}

bool RookPawnModel::validateMove(BoardPosition positionToMove, BasePawnModel \*pawnOnPositionToMove, BoardPosition \*requestedActivePawnPosition) {

if (pawnOnPositionToMove && pawnOnPositionToMove->owner == this->owner) {

return false;

}

return validateVerticalOrHorizontalMove(positionToMove, pawnOnPositionToMove);

}

**Файл utils.cpp**

#include "utils.h"

#include <QBrush>

#include <QLabel>

#include <QGraphicsProxyWidget>

void Utils::setBackgroundColor(QColor color, QAbstractGraphicsShapeItem \*item) {

QBrush brush;

brush.setStyle((Qt::SolidPattern));

brush.setColor(color);

item->setBrush(brush);

}

void Utils::setImage(QString imagePath, QGraphicsRectItem \*item) {

QPixmap image(imagePath);

QLabel \*imageLabel = new QLabel();

QGraphicsProxyWidget \*pMyProxy = new QGraphicsProxyWidget(item);

imageLabel->setPixmap(image);

imageLabel->setAttribute(Qt::WA\_TranslucentBackground);

pMyProxy->setWidget(imageLabel);

}

QGraphicsTextItem\* Utils::createTextItem(QString title, int fontSize, QColor textColor, QGraphicsItem \*parent) {

QGraphicsTextItem \*textItem = new QGraphicsTextItem(title, parent);

QFont titleFont("avenir", fontSize);

textItem->setDefaultTextColor(textColor);

textItem->setFont(titleFont);

return textItem;

}