Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Игра «Шахматы»

БГУИР КП 1–40 02 01 205 ПЗ

Студент: группы 250502,   
Бригадир А. С.

Руководитель: ассистент каф. ЭВМ Богдан Е. В.

Минск 2023

Учреждение образования

«Белорусский Учреждение образования

«Белорусский государственный университет информатики   
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ

Заведующий кафедрой

*––––––––––––––––––––––––*

(подпись)

––––––––––––––––––2023 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту     *Бригадир Анне Сергеевне –––––––––––––*

Тема проекта *Игра «Шахматы»* \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Срок сдачи студентом законченного проекта––––*15 декабря 2023 г.–––*

3. Исходные данные к проекту*-пЯзык программирования – С++, среда разработки – Qt-Creator ------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------*

*––––––––––––––––––––––––––––––––––––––––––––––––––––––           —–––*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

1. Лист задания.

2. Введение.

3. Обзор литературы.

3.1. Анализ существующих аналогов

3.1. Требования к работе программы

4. Системное проектирование.

4.1. Приветственное меню приложения.

4.2. Главное игровое окно.

4.3. Конечное окно.

5. Функциональное проектирование.

5.1. Структура входных и выходных данных.

5.2. Разработка диаграммы классов.

5.3. Описание классов.

6. Разработка программных модулей.

6.1. Алгоритм обработки нажатия на фигуру.

6.2. Алгоритм обработки нажатия на ячейку поля.

7. Результаты работы.

8. Заключение

9. Литература

10. Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

*1. Диаграмма классов.*

*2. Схема алгоритма обработки нажатия на фигуру*

*3. Схема алгоритма обработки нажатия на ячейку поля*

6. Консультант по проекту (с обозначением разделов проекта) Е. В. Богдан

7. Дата выдачи задания –––––*15.09.2023г.––––––––––––––––––––––   –*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

*1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала – 15 %;––––––––––––––––––––––––––––*

*разделы 2, 3 – 10 %;–––––––––––––––––––––––––––––––––––––*

*разделы 4 к –20 %;–––––––––––––––––––––––––––––––––––––––*

*разделы 5 к – 35 %;––––––––––––––––––––––––––––––––––––––*

*раздел 6,7,8 – 5 %;–––––––––––––––––––––––––––––––––––––––*

*раздел 9 к – 5%;–––––––––––––––––––––––––––––––––––––––––*

*оформление пояснительной записки и графического материала к 15.12.23 – 10 %*

*Защита курсового проекта с 21.12 по 28.12.23г.–––––––––––––––––––––––––*

РУКОВОДИТЕЛЬ Е.В. Богдан

(подпись)

Задание принял к исполнению *–––––––\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_––*

(дата и подпись студента)

**СОДЕРЖАНИЕ**

ВВЕДЕНИЕ 5

1 ПОСТАНОВКА ЗАДАЧИ 6

2 ОБЗОР ЛИТЕРАТУРЫ 11

* 1. Анализ существующих аналогов 11
     1. Chess.com 11
     2. Chess for windows 12
  2. Требования к работе программы 12

3 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ 14

3.1 Приветственное меню приложения 14

3.2 Главное игровое окно 14

* 1. Конечное окно 15

1. ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ 16
   1. Входные и выходные данные 16
   2. Разработка диаграммы классов 16
   3. Описание классов.. 16
      1. Класс MainWindow 16
      2. Класс Button 17
      3. Класс Game 17
      4. Класс ChessBoard 18
      5. Класс ChessBox 19
      6. Класс Chesspiece 19
      7. Класс Bishop 20
      8. Класс King 21
      9. Класс Knight 21
      10. Класс Pawn 21
      11. Класс Queen 21
      12. Класс Rook 22
2. РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ 23
   1. Алгоритм обработки нажатия на фигуру 23
   2. Алгоритм обработки нажатия на ячейку поля 23
3. РЕЗУЛЬТАТЫ РАБОТЫ 25

ЗАКЛЮЧЕНИЕ 28

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 29

ПРИЛОЖЕНИЕ А 30

ПРИЛОЖЕНИЕ Б 31

ПРИЛОЖЕНИЕ В 32

ПРИЛОЖЕНИЕ Г 33

ПРИЛОЖЕНИЕ Г 64

**ВВЕДЕНИЕ**

C++ является одним из наиболее популярных языков программирования, который широко используется для разработки приложений, включая системное программное обеспечение, игры, веб-приложения и многое другое. Он был разработан в 1979 году Бьярном Страуструпом как расширение языка программирования C и предоставляет разработчикам мощные инструменты для создания различных типов приложений. Он поддерживает процедурное, объектно-ориентированное и обобщенное программирование, что делает его очень гибким и универсальным языком.

ООП – это парадигма программирования, которая позволяет организовать код в виде объектов, которые взаимодействуют друг с другом. Основными принципами ООП являются наследование, полиморфизм и инкапсуляция.

Наследование позволяет создавать иерархию классов, где дочерние классы наследуют свойства и методы от родительских классов. Это позволяет повторно использовать код и создавать более абстрактные и специализированные классы.

Полиморфизм позволяет использовать один и тот же интерфейс для работы с разными типами объектов. Это достигается с помощью виртуальных функций и переопределения методов в дочерних классах.

Инкапсуляция позволяет скрыть детали реализации класса и предоставить только необходимый интерфейс для работы с ним. Это достигается с помощью модификаторов доступа, таких как public, private и protected.

Qt – это библиотека классов C++ и набор инструментального программного обеспечения для создания кросс-платформенных приложений с графическим интерфейсом (GUI).

Qt позволяет запускать написанное с его помощью программное обеспечение в большинстве современных операционных систем путём простой компиляции программы для каждой системы без изменения исходного кода. Включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, начиная от элементов графического интерфейса и заканчивая классами для работы с сетью, базами данных и XML. Является полностью объектно-ориентированным, расширяемым и поддерживающим технику компонентного программирования.

# 1 ПОСТАНОВКА ЗАДАЧИ

Тема курсового проекта была выбрана в первую очередь для получения знаний в области разработки десктопных приложений с применением ООП, для изучения возможностей Qt – библиотеки классов C++ и набора инструментального программного обеспечения для создания кросс-платформенных приложений с графическим интерфейсом.

Правила игры «Шахматы» включают в себя различные алгоритмы совершения хода, зависящие от вида фигуры, ее положения на доске, возможности создать ситуацию угрозы фигуре противника, конечной целью является взятие короля. Всего в шахматах есть 6 различных видов фигур.

Король - самая важная фигура, но при этом и одна из самых слабых. Король может ходить только на одно поле в любом направлении: вверх, вниз, в стороны и по диагонали. Король никогда не может вставать на битое поле (где его может взять фигура соперника). Когда король атакован другой фигурой, это называется "шах". На рисунках 1.1(а) и 1.1(б) представлены иконки фигур белого и черного короля.

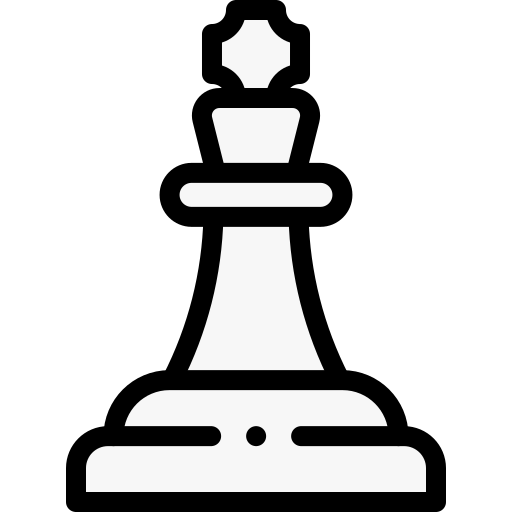


Рисунок 1.1(а) – Иконка белого короля.

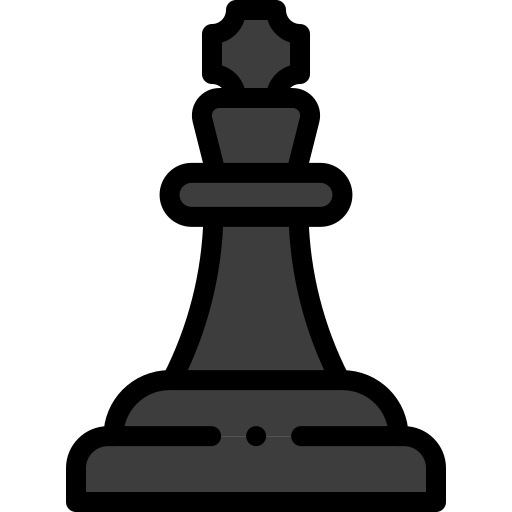


Рисунок 1.1(б) – Иконка черного короля.

Ферзь – самая сильная фигура. Он может ходить по прямой в любом направлении – вперёд, назад, в стороны или по диагонали на любое число полей, но при этом он не может перепрыгивать через другие фигуры. Если ферзь или любая другая фигура берёт фигуру соперника, ход заканчивается. На рисунках 1.2(а) и 1.2(б) представлены иконки фигур белого и черного ферзя.

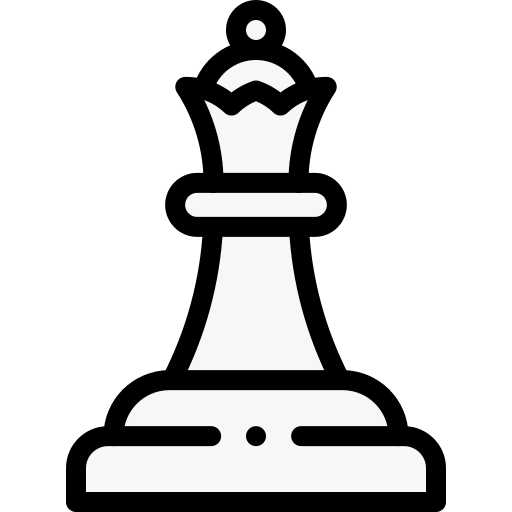


Рисунок 1.2(а) – Иконка белого ферзя.



Рисунок 1.2(б) – Иконка черного ферзя.

Ладья может ходить на любое число полей, но только вперёд, назад и в стороны (не по диагонали). На рисунках 1.3(а) и 1.3(б) представлены иконки фигур белой и черной ладьи.

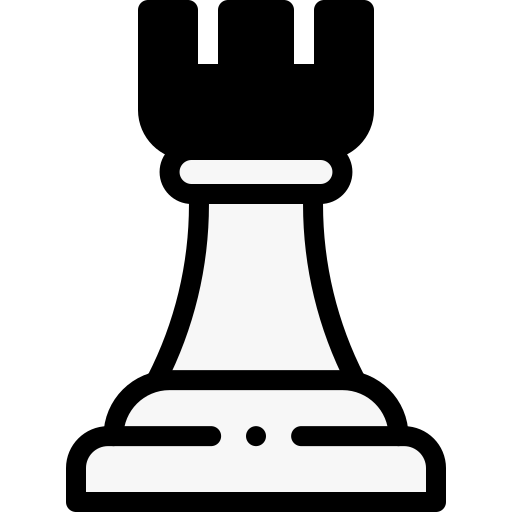


Рисунок 1.3(а) – Иконка белой ладьи.

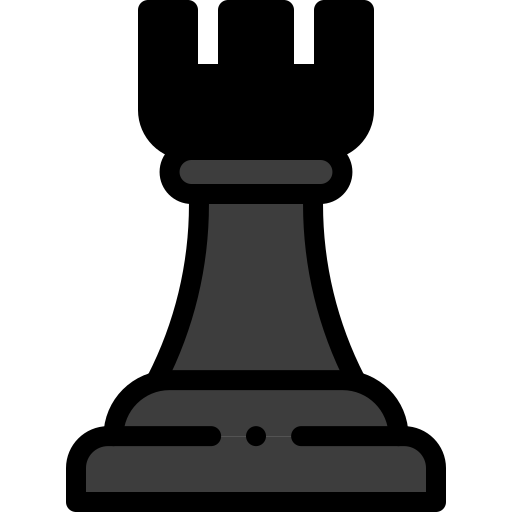


Рисунок 1.3(б) – Иконка черной ладьи.

Слон может ходить на любое число полей, соединенных углами по диагонали. Каждому слону доступна только половина полей доски (одного цвета, белого или черного). На рисунках 1.4(а) и 1.4(б) представлены иконки фигур белого и черного слона.



Рисунок 1.4(а) – Иконка белого слона.



Рисунок 1.4(б) – Иконка черного слона.

Кони ходят иначе, чем другие фигуры - на два поля в одном направлении и далее на одно поле под углом 90 градусов, буквой "Г". Конь - единственная фигура, способная перепрыгивать через другие фигуры. На рисунках 1.5(а) и 1.5(б) представлены иконки фигур белого и черного коня.



Рисунок 1.5(а) – Иконка белого коня.



Рисунок 1.5(б) – Иконка черного коня.

Пешка - необычная фигура, она ходит и берёт по-разному: ходить пешка может лишь вперёд, а брать - лишь по диагонали. Пешка может передвигаться только на одно поле за ход, но еще не ходившая пешка может пойти вперёд на одно или два поля. Пешка может брать только по диагонали на одно поле перед собой. Пешка не может ходить или брать назад. Если другая фигура находится прямо перед пешкой, та не может сделать ход вперёд и не может взять эту фигуру. На рисунках 1.6(а) и 1.6(б) представлены иконки фигур белой и черной ладьи.

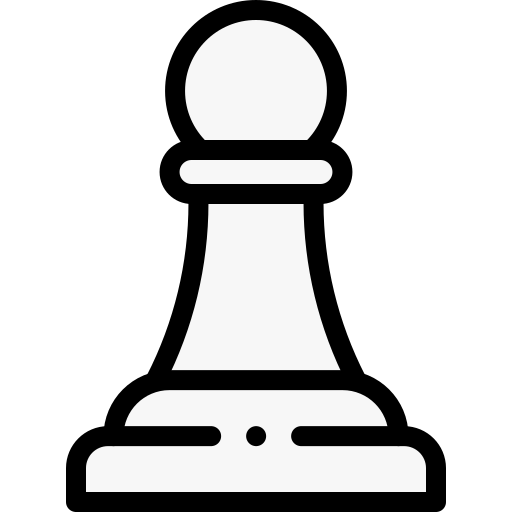


Рисунок 1.6(а) – Иконка белой пешки.

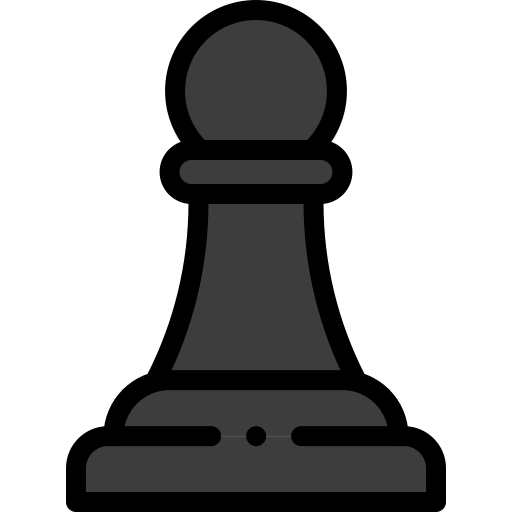


Рисунок 1.6(б) – Иконка черной пешки.

Цель игры - поставить мат королю соперника. На доске мат, когда один из игроков не может защититься от шаха. Защититься можно отступив на другое поле, закрывшись от шаха другой фигурой или взяв фигуру, напавшую на короля.

**2 ОБЗОР ЛИТЕРАТУРЫ**

**2.1 Анализ существующих аналогов**

Для создания собственного работающего приложения, необходимо иметь представление об уже существующих аналогах.

* + 1. **Chess.com**

Chess.com – сайт для игры в шахматы онлайн, форум, шахматная социальная сеть, которая так же имеет свое десктопное приложение, которое позволяет играть в реальном времени с другими игроками, обучаться стратегиям и тактике, просматривать профессиональные партии и многое другое. На рисунке 2.1 показана часть интерфейса сайта. При нажатии на фигуру отображаются возможные для нее ходы. Есть возможность сыграть с искусственным интеллектом с разными уровнями сложности.



Рисунок 2.1 – Скриншот с сайта Chess.com.

* + 1. **Chess for windows**

Игра «Шахматы» из официального приложения Microsoft Store. В данном приложении так же есть возможность сыграть с искусственным интеллектом разного уровня, выбрать цвет фигур пользователя в начале игры. Есть такая опция, как подсказка, которая делает следующий ход вместо игрока.

****

Рисунок 2.2 – Скриншот приложения «Chess for windows».

* 1. **Требования к работе программы**

После рассмотрения уже существующих аналогов игры «Шахматы» можно выделить основные задачи, которые необходимо выполнить для реализации собственного приложения. Для разработки десктопной игры «Шахматы» необходимо написать игровую логику шахмат и создать пользовательский интерфейс, который включает в себя саму доску с фигурами, отображение возможных ходов для каждый фигуры, очередь хода.

Для реализации игровой логики шахмат необходимо создать классы для шахматной доски и фигур, определить методы для проверки возможных ходов, наличия шаха, передвижения фигур.

Для написания приложения был выбран язык С++, так как необходимо использовать объектно–ориентированную парадигму, что делает удобным создание и взаимодействие нескольких окон. Полная информация о языке приведена в источниках [1] и [2].

Для выполнения графического интерфейса была выбрана Qt – библиотека классов C++ и набор инструментального программного обеспечения для создания кросс-платформенных приложений с графическим интерфейсом. Полная информация о QT приведена в источниках [3] и [4].

**3 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ**

Весь функционал приложения можно разбить на блоки для упрощения разработки и конечной реализации.

* 1. **Приветственное меню приложения**

Данное окно включает визуальное отображение приветствия пользователя, возможность начать игру или выйти из приложения.

Для создания и отображения графики главного меню можно использовать Qt Designer – графический инструмент для создания пользовательских интерфейсов, который предоставляет широкий выбор готовых виджетов и макетов, которые легко настраиваются и располагаются для создания нужного пользовательского интерфейса. Базовым классом для этого окна является класс Qt QMainWindow, который представляет собой виджет для создания основного окна приложения. Он обеспечивает основную структуру и функциональность для главного окна приложения, включая меню, панели инструментов, статусную строку и т. д.

* 1. **Главное игровое окно**

Главное игровое окно является основным в данном приложении. Оно включает в себя саму доску с фигурами, отображение активного хода, отображение фигур, которые выбыли из игры, предупреждение о возможном шахе.

Поскольку данное окно имеет большое количество различных объектов, свойства которых необходимо менять в зависимости от ситуации, для реализации был выбран базовый класс QGraphicsScene. С помощью данного класса можно добавлять различные графические элементы, такие как прямоугольники, эллипсы, линии, текст и изображения, что и необходимо при отрисовке интерфейса приложения. QGraphicsScene также обеспечивает удобные методы для обработки событий, таких как нажатие кнопок мыши, перемещение указателя мыши и другие пользовательские взаимодействия. Данный класс наследуется для отображения различных полей и примитивных объектов, не требующих установки изображения, такие как ячейки доски, боковые поля для выбывших фигур, надписи.

Для отображения фигур шахмат был выбран базовый класс QGraphicsPixmapItem, который представляет собой графический элемент, предназначенный для отображения изображений на сцене QGraphicsScene. Он является подклассом класса QGraphicsItem и предоставляет удобные методы для работы с изображениями.

* 1. **Конечное окно**

Конечное окно предназначено для уведомления пользователя о

победившей стороне. После окончания игры отображается начальное окно, где вместо кнопки «play» отображается надпись с победившей стороной.

Необходимая надпись устанавливается при поступлении сигналов игрового окна, после чего выполняются соответствующие слоты начального окна.

**4 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ**

В данном разделе описываются входные и выходные данные программы, диаграмма классов, а также приводится описание используемых классов и их методов.

**4.1 Входные и выходные данные**

Пользователь начинает игру нажатием на кнопку, выбирает ходы фигур путем нажатия клавиши мыши, выходными данными являются передвижения фигур, отображение возможных ходов, текстовые уведомления пользователя о выигрыше и проигрыше.

**4.2 Разработка диаграммы классов**

Диаграмма классов представлена в Приложении А.

* 1. **Описание классов**
     1. **Класс MainWindow**

Собственный класс MainWindow наследует класс QMainWindow – это класс в библиотеке Qt, который содержит основные элементы для создания и управления главным окном приложения, в данном случае пользовательское меню.

Поля:

MainWindow \*ui; – поле дизайн-макета главного окна

Методы:

MainWindow(QWidget \*parent = nullptr); – конструктор класса, который устанавливает в окне дизайн-макет

~MainWindow(); – деструктор класса, который очищает в окне установленный дизайн-макет

Слоты:

void on\_buttonEXIT\_clicked(); – функция закрытия окна при поступлении соответствующего сигнала

void on\_buttonPVP\_clicked(); – функция открытия игрового окна при поступлении соответствующего сигнала

void whiteWON(); – функция вызова финального окна при поступлении соответствующего сигнала

void blackWON(); – функция вызова финального окна при поступлении соответствующего сигнала

* + 1. **Класс Button**

Класс Button **–** класс кнопки, который наследует базовые классы QT QObject и QGraphicsRectItem.

Поля:

QGraphicsTextItem\* text; – надпись внутри кнопки

Методы:

Button(QString name, QGraphicsItem\* parent = NULL); – конструктор, который создает объект прямоугольника определенного размера и цвета, задает положение текста внутри

void setFont(QFont const textFont); – установка различных параметров текста с помощью базового класса QT QFont

void mousePressEvent(QGraphicsSceneMouseEvent\* event); – обработка нажатий мыши

void hoverEnterEvent(QGraphicsSceneHoverEvent\* event); – обработка наведения курсора мышки на объект

void hoverLeaveEvent(QGraphicsSceneHoverEvent\* event); – обработка прекращения наведения курсора мышки на объект

Сигналы:

void clicked(); – обработка нажатия мышки на объект

* + 1. **Класс Game**

Класс Game **–** класс основного цикла игры, в котором происходит начальная отрисовка сцены, которая включает шахматную доску, фигуры, поля для выбывших фигур, отображение очереди хода и наличия шаха. Является дочерним от базового класса QT QGraphicsScene.

Поля:

QGraphicsScene \*gameScene; – графическая сцена игрового окна

QList <ChessPiece \*> whiteDead; – список объектов класса

ChessPiece, хранящий все выбывшие белые фигуры

QList <ChessPiece \*> blackDead; – список объектов класса

ChessPiece, хранящий все выбывшие черные фигуры

QGraphicsRectItem \* deadHolder; – графический объект боковых

полей для отображения выбывших фигур и основного поля шахматной доски

QString turn; – очередь хода

ChessBoard \*chess; – графический объект класса ChessBoard

QList <QGraphicsItem \*> listG; – список графических объектов

QGraphicsTextItem \* turnDisplay; – графический текстовый объект для отображения очереди хода

ChessBox \*collection[8][8]; –массив объектов класса ChessBox,

хранящий все ячейки доски

QList <ChessPiece \*> alivePiece; – список объектов класса

ChessPiece оставшихся фигур

ChessPiece \*pieceToMove; – объект класса ChessPiece, активная фигура

Методы:

Game(QWidget \*parent = 0); – конструктор класса, который устанавливает в окне такие графические объекты, как сама графическая сцена, текстовые объекты очереди хода и наличия шаха

void drawChessBoard(); – метод добавления шахматной доски,

боковых панелей для выбывших фигур

void displayDeadWhite(); – отображение списка объектов класса

ChessPiece выбывших белых фигур

void displayDeadBlack(); – отображение списка объектов класса

ChessPiece выбывших черных фигур

void addToScene(QGraphicsItem \*item); – метод для добавления в

графическую сцену объекта класса QGraphicsItem

void removeFromScene(QGraphicsItem \*item); – метод для

удаления из графической сцены объекта класса QGraphicsItem

QString getTurn(); – метод, возвращающий очередь хода

void setTurn( QString value); – метод, устанавливающий очередь

хода

void changeTurn(); – смена хода на противоположную

void drawDeadHolder(int x, int y,QColor color); – метод для

отрисовки зон для выбывших фигур и шахматной доски

void gameOver(); – окончание игры

void removeAll(); – прекращение отображения всех объектов

void start(); – начало игры, добавление в сцену всех необходимых

объектов, начало их отображения, добавление всех фигур

* + 1. **Класс ChessBoard**

Класс ChessBoard **–** класс, необходимый для управления и систематизации объектов доски, управления ими, создания единой системы из ячеек доски и фигуры, находящейся в этой ячейке.

Поля:

QList <ChessPiece\*> white; – список белых фигур

QList <ChessPiece\*> black; – список белых фигур

Методы:

ChessBoard();– конструктор класса, который выполняет заполнение списков всех черных и белых фигур методами setUpWhite() и setUpBlack() соответственно

void drawBoxes(int x, int y); – отрисовка всей шахматной доски, состоящей из объектов класса ChessBox

void setUpWhite(); – заполнение списка всех черных фигур

void setUpBlack(); – заполнение списка всех белых фигур

void addChessPiece(); – установка в необходимые ячейки доски соответствующих фигур из списков

* + 1. **Класс ChessBox**

Класс ChessBox **–** класс графического отображения ячейки шахматной доски, описывающий взаимодействие пользователя с фигурами и ячейками, а также связь между находящейся в ячейке фигурой и самой ячейкой. Наследует базовый класс QT QGraphicsRectItem.

Поля:

QColor originalColor; – начальный цвет ячейки

bool hasChessPiece; – наличие фигуры в ячейке

QString chessPieceColor; – цвет фигуры в ячейке

int rowLoc; – номер строки доски

int colLoc; – номер столбца доски

ChessPiece \* currentPiece; – указатель на фигуру в ячейке

QBrush brush; – объект базового класса QT QBrush с графическими параметрами ячейки

Методы:

ChessBox(QGraphicsItem\* parent = 0); – конструктор класса, который создает объект ячейки поля и устанавливает начальные значения полей объекта

~ChessBox(); – деструктор класса

void mousePressEvent(QGraphicsSceneMouseEvent\* event); – обработка нажатия мыши на ячейку

void setColor(QColor color); – установка цвета ячейки

void placePiece(ChessPiece\* piece); – установка фигуры в ячейке

void resetOriginalColor(); – восстановление изначального цвета ячейки

void setOriginalColor(QColor value); – установка и восстановление изначального цвета ячейки

void setChessPieceColor(QString value); – установка цвета фигуры в ячейке

QString getChessPieceColor(); – получение цвета фигуры в ячейке

* + 1. **Класс ChessPiece**

Класс ChessPiece **–** базовыйкласс игровой фигуры. Включает в себя все необходимые для наследования функции. Наследует базовый класс QT QGraphicsPixmapItem.

Поля:

bool firstMove; – первый ход фигуры

ChessBox \*currentBox; – активная ячейка поля

QString side; – ход игры

bool isPlaced; – находится ли фигура в ячейке игрового поля

QList <ChessBox \*> location; – список указателей на объекты класса

ячейки поля, хранящий все возможные ходы фигуры

Методы:

ChessPiece(QString team = "",QGraphicsItem \*parent = 0); –

конструктор класса, который задает цвет игровой фигуры, начальные параметры первого хода и установки в ячейке

void mousePressEvent(QGraphicsSceneMouseEvent\* event); – обработка нажатий мыши, выбор фигуры, просчет ее возможных ходов, снятие выделения в том случае, если уже есть другая выделенная фигура.

virtual void setImage() = 0; – установка изображения фигуры

bool getIsPlaced(); – возвращает значение isPlaced, находится ли фигура в ячейке игрового поля

void setIsPlaced(bool value); – установка значения isPlaced, находится ли фигура в ячейке игрового поля

QList <ChessBox\*> moveLocation(); – возвращает список

указателей на объекты класса ячейки поля, возможных для хода

virtual void moves() = 0; – метод полной обработки хода фигуры

void setCurrentBox(ChessBox \*box); – установка указателя на ячейку поля, где находится фигура

ChessBox \*getCurrentBox(); – возвращает указатель на ячейку поля, где находится фигура

QString getSide(); – возвращает значение цвета фигуры

void setSide( QString value); – устанавливает значение цвета фигуры

void decolor(); – устанавливает начальный цвет ячеек поля после снятия выделения с фигуры

* + 1. **Класс Bishop**

Класс Bishop **–** класс фигуры слона, наследует базовый класс фигуры ChessPiece.

Методы:

Bishop(QString team, QGraphicsItem\* parent = 0); –

конструктор класса, который задает цвет игровой фигуры, начальные параметры первого хода и установки в ячейке

void setImage(); – установка изображения фигуры

void moves(); – метод полной обработки хода фигуры

* + 1. **Класс King**

Класс King **–** класс фигуры короля, наследует базовый класс фигуры ChessPiece.

Методы:

King(QString team, QGraphicsItem\* parent = 0); –

конструктор класса, который задает цвет игровой фигуры, начальные параметры первого хода и установки в ячейке

void setImage(); – установка изображения фигуры

void moves(); – метод полной обработки хода фигуры

void findUnSafeLocation(); – поиск ячеек, при которых король становится в позицию шаха

* + 1. **Класс Knight**

Класс Knight **–** класс фигуры коня, наследует базовый класс фигуры ChessPiece.

Методы:

Knight(QString team, QGraphicsItem\* parent = 0); –

конструктор класса, который задает цвет игровой фигуры, начальные параметры первого хода и установки в ячейке

void setImage(); – установка изображения фигуры

void moves(); – метод полной обработки хода фигуры

* + 1. **Класс Pawn**

Класс Pawn **–** класс фигуры пешки, наследует базовый класс фигуры ChessPiece.

Методы:

Pawn (QString team, QGraphicsItem\* parent = 0); –

конструктор класса, который задает цвет игровой фигуры, начальные параметры первого хода и установки в ячейке

void setImage(); – установка изображения фигуры

void moves(); – метод полной обработки хода фигуры

* + 1. **Класс Queen**

Класс Queen **–**класс фигуры ферзя, наследует базовый класс фигуры ChessPiece.

Методы:

Queen (QString team, QGraphicsItem\* parent = 0); –

конструктор класса, который задает цвет игровой фигуры, начальные параметры первого хода и установки в ячейке

void setImage(); – установка изображения фигуры

void moves(); – метод полной обработки хода фигуры

* + 1. **Класс Rook**

Класс Rook **–** класс фигуры ладьи, наследует базовый класс фигуры ChessPiece.

Методы:

Rook (QString team, QGraphicsItem\* parent = 0); –

конструктор класса, который задает цвет игровой фигуры, начальные параметры первого хода и установки в ячейке

void setImage(); – установка изображения фигуры

void moves(); – метод полной обработки хода фигуры

**5 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ**

В данном разделе рассмотрены описания алгоритмов, используемых в программе.

**5.1 Алгоритм обработки нажатия на фигуру**

Алгоритм обработки нажатия на фигуру представлен в методе mousePressEvent(QGraphicsSceneMouseEvent \*event) класса ChessPiece.

Шаг 1. Начало метода.

Шаг 2. Если эта фигура является выделенной, то перейти к Шагу 4, иначе перейти к Шагу 7.

Шаг 3. Вернуть изначальную окраску ячейке, хранящей данную фигуру.

Шаг 4. Вернуть изначальную окраску ячейкам возможных ходов.

Шаг 5. Обнулить переменную-указателю на активную фигуру.

Шаг 6. Перейти к Шагу 15.

Шаг 7. Если данная фигура является фигурой противоположного от хода цвета при отсутствии уже выделенной фигуры либо эта фигура является выбывшей из игры, то перейти к Шагу 15, иначе перейти к Шагу 8.

Шаг 8. Если выделенной фигуры нет, то перейти к Шагу 9, иначе перейти к Шагу 13.

Шаг 9. Данная фигура становится выделенной.

Шаг 10. Сменить цвет ячейки выделенной фигуры.

Шаг 11. Вычислить все возможные ходы.

Шаг 12. Перейти к Шагу 15.

Шаг 13. Если цвет фигуры не совпадает с цветом хода, то перейти к Шагу 14, иначе перейти к Шагу 15.

Шаг 14. Выполнить событие нажатия мыши для текущей ячейки.

Шаг 15. Завершение метода.

**5.2 Алгоритм обработки нажатия на ячейку поля**

Алгоритм обработки нажатия на ячейку поля представлен в методе mousePressEvent(QGraphicsSceneMouseEvent \*event) класса ChessBox.

Шаг 1. Начало метода.

Шаг 2. Если фигура в этой ячейке является выделенной и находится в игре, то перейти к Шагу 3, иначе перейти к Шагу 5.

Шаг 3. Выполнить событие нажатия мыши для текущей ячейки.

Шаг 4. Перейти к Шагу 29.

Шаг 5. Если уже есть выделенная фигура, то перейти к Шагу 6, иначе перейти к шагу 27.

Шаг 6. Если фигура в ячейке является фигурой того же цвета, что и цвет хода, то перейти к Шагу 29, иначе перейти к Шагу 7.

Шаг 7. Создать список ячеек, возможных для хода выделенной фигуры.

Шаг 8. Объявить переменнуюи check для проверки наличия данной ячейки в списке возможных ходов выделенной фигуры.

Шаг 9. Начало цикла проверки, инициализировать переменную i равную 0 и переменной n равную размеру массива списка возможных ходов.

Шаг 10. Если i меньше n, выполнить Шаг 11, иначе перейти на Шаг 12.

Шаг 11. Если элемент списка с индексом i является данной ячейкой, увеличить значение переменной check на 1.

Шаг 12. Если переменная check имеет значение 0, то есть ячейка не находится в списке возможных ходов, перейти к Шагу 29, иначе перейти к Шагу 13.

Шаг 13. Вернуть изначальную окраску ячейке активной фигуры.

Шаг 14. Присвоить полю первого хода активной фигуры значение false.

Шаг 15. Если в данной ячейке находится фигура, то перейти к шагу 16, иначе перейти к шагу 19.

Шаг 16. Присвоить полю нахождения в ячейке в выбывшей фигуре значение false.

Шаг 17. Обнулить указатель на ячейку расположения выбывшей фигуры.

Шаг 18. Добавить выбывшую фигуру в список выбывших фигур.

Шаг 19. Присвоить полю хранения фигуры в старой ячейке активной фигуры значение false.

Шаг 20. Обнулить указатель на хранящуюся фигуру в старой ячейке активной фигуры.

Шаг 21. Вернуть изначальную окраску старой ячейке активной фигуры.

Шаг 22. Разместить активную фигуру в данное поле.

Шаг 23. Обнулить указатель на активную фигуру.

Шаг 24. Сменить ход.

Шаг 25. Выполнить проверку на наличие шаха.

Шаг 26. Если в этой ячейке есть фигура, то перейти к Шагу 3, иначе перейти к Шагу 29.

Шаг 27. Передача нажатия кнопки мыши в метод mousePressEvent(QGraphicsSceneMouseEvent \*event) класса ChessPiece.

Шаг 28. Переход к Шагу 29.

Шаг 29. Завершение метода.

**6 РЕЗУЛЬТАТЫ РАБОТЫ**

При запуске программы отображается начальное окно (Рисунок 6.1). Пользователь может нажать на кнопку «play», чтобы начать игру, либо на кнопку «exit» для выхода из приложения.



Рисунок 6.1 – Начальное окно.

После нажатия кнопки начала игры данное окно сворачивается и открывается основное игровое окно (Рисунок 6.2). Для того, чтобы выбрать фигуру для хода, необходимо нажать на нее кнопкой мыши, после чего, в случае удачного выделения, ячейка с выделенной фигурой подсветится темно-зеленым, также подсветятся и все возможные для этой фигуры ходы. Для отображения разных типов хода используются разные цвета подсветки: для обычного хода ярко-зеленый, желтым подсвечиваются фигуры, который можно побить, фиолетовым подсвечивается король, если ему угрожает какая-либо фигура. Это можно увидеть на Рисунках 6.3(*а*) и 6.3(*б*).

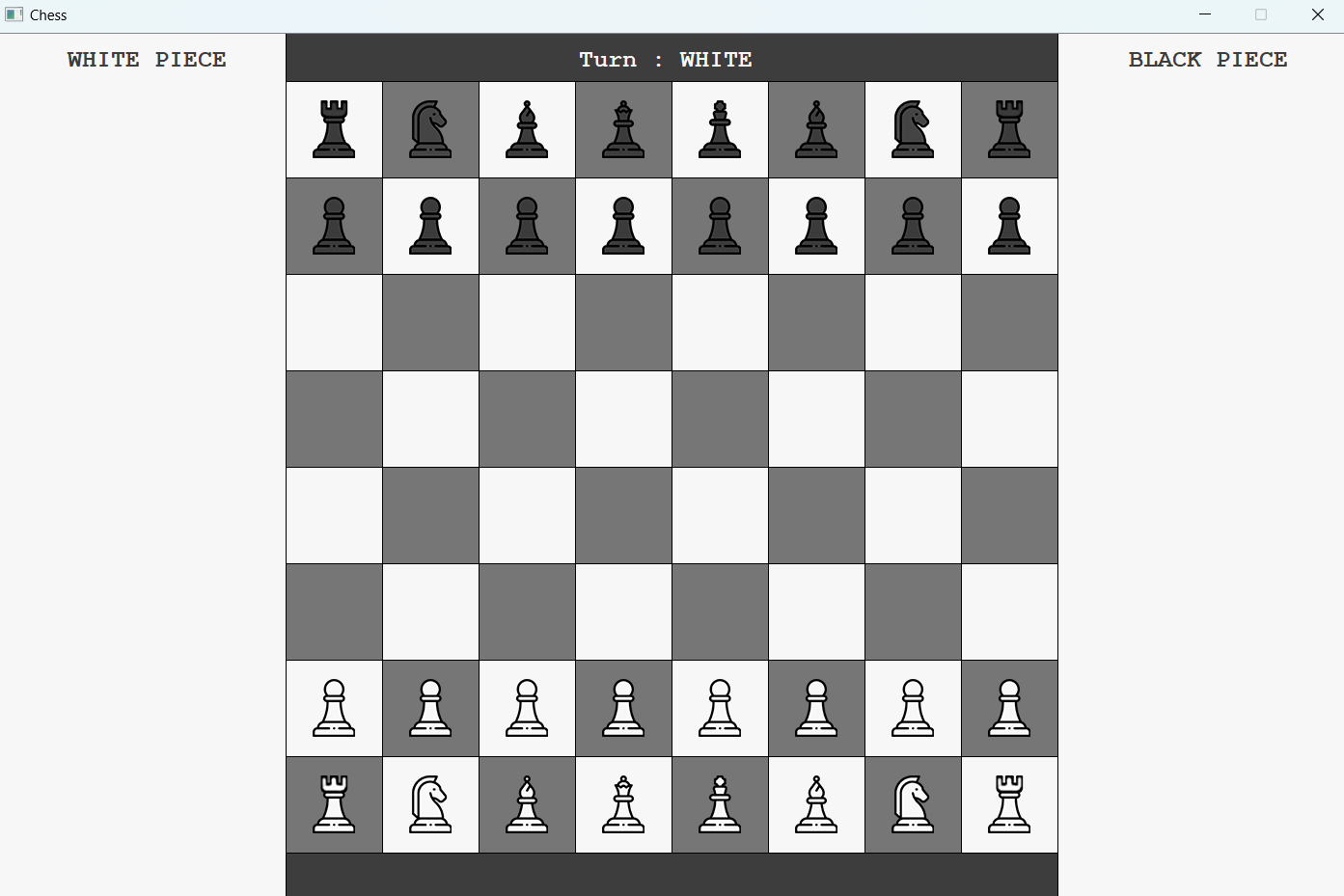


Рисунок 6.2 – Начальное игровое окно.

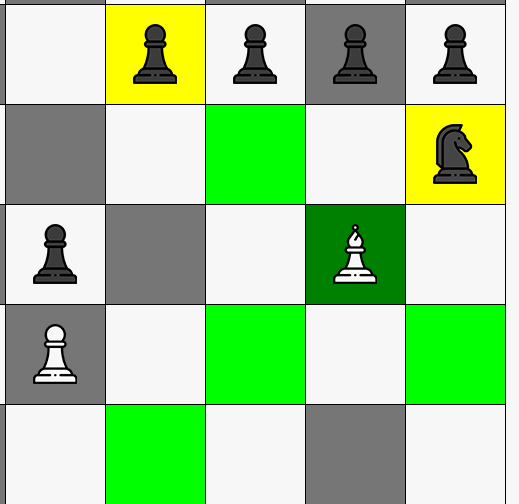
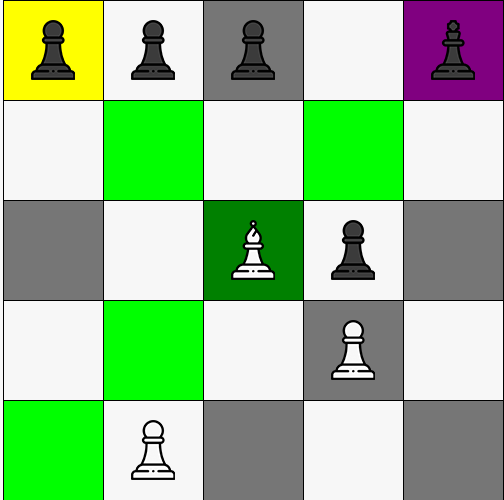
*а) б)*

Рисунок 6.3 – Подсветка разных типов хода: *а* – подсветка фигур, которых можно побить; *б* – подсветка короля в опасном положении.

После взятия фигуры она перемещается в боковую панель выбывших фигур соответствующего цвета. В случае, если королю угрожает фигура, снизу доски отображается надпись «Check!» (Рисунок 6.4).

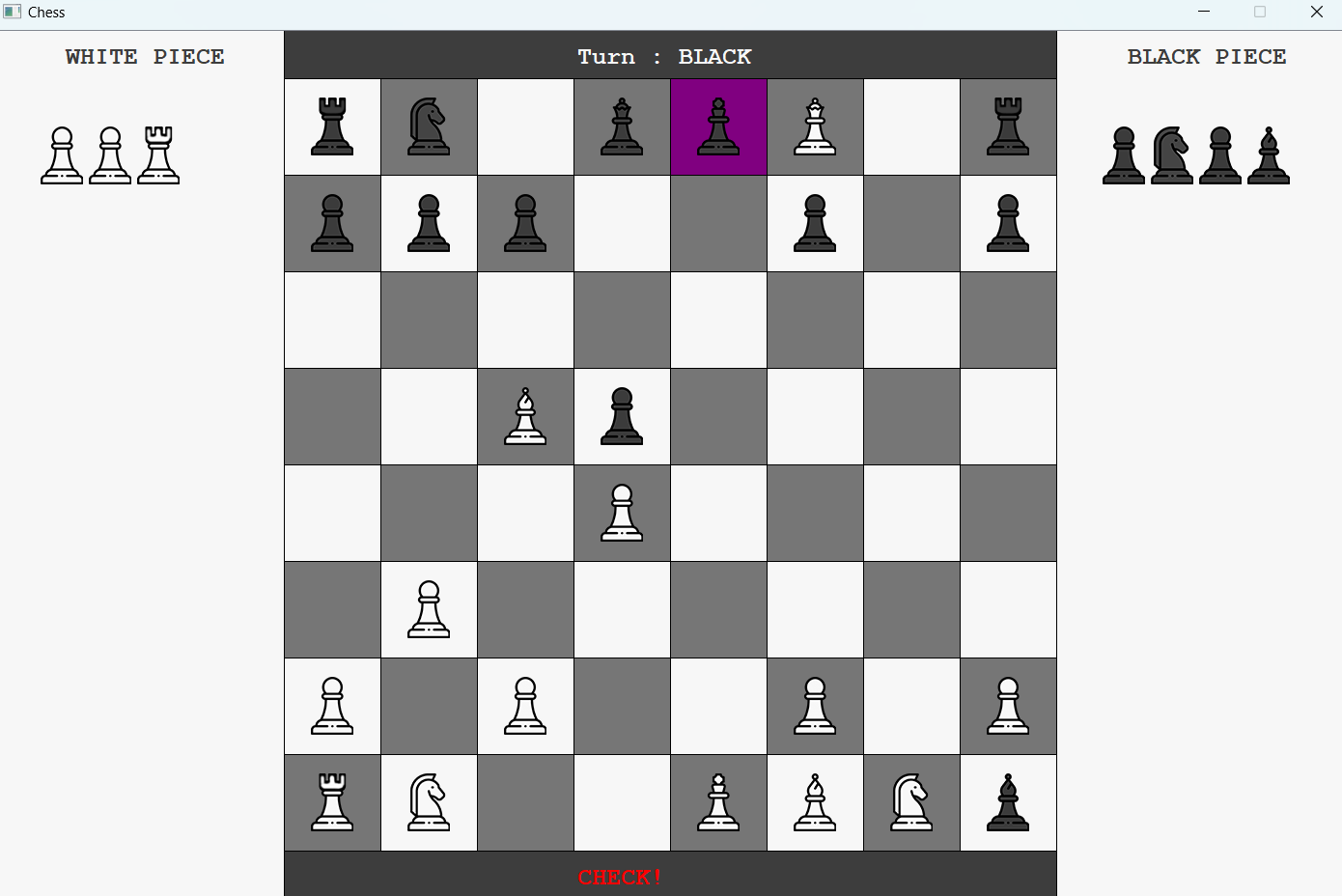
****

Рисунок 6.4 – Надпись «Check!».

Если король был побит, главное игровое окно закрывается, вновь появляется начальное окно, где отображается информация о выигравшей стороне (Рисунок 6.5). Игра завершена.

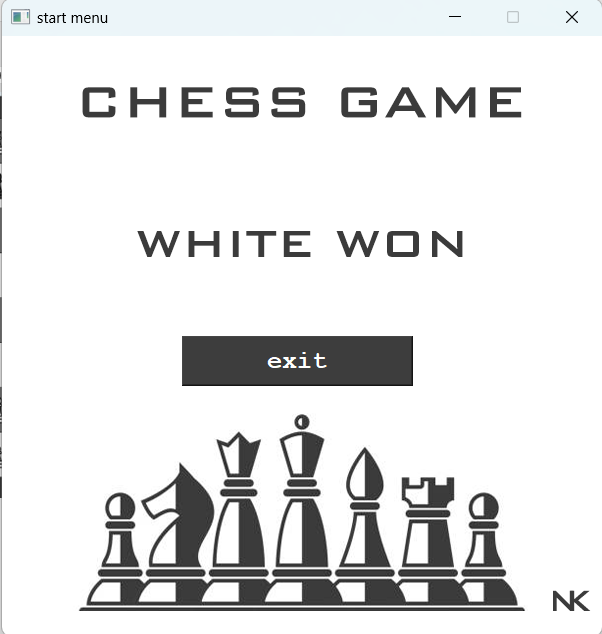
****

Рисунок 6.5 – Завершение игры.

**ЗАКЛЮЧЕНИЕ**

Данный курсовой проект представляет собой готовое приложение, которое обеспечивает базовый функционал для игры в шахматы, что делает его отличным выбором как для начинающих игроков, так и для опытных шахматистов.

В процессе работы достигнуты поставленные цели, а именно был разработан алгоритм перемещения фигур, обработки ходов и проверки наличия шаха. Также был создан графический интерфейс приложения, выполненный в минималистичном стиле, что делает его удобным и легким в использовании.

За время создания данного курсового проекта были приобретены и улучшены навыки программирования на QT, а также понимание логики игры в шахматы. Для работы с графикой были использованы классы QGraphicsView, QGraphicsScene и QGraphicsPixmapItem, которые обеспечивают возможности для отображения графических сцен (QGraphicsScene), взаимодействия с различными элементами сцены.

Создание курсового проекта развило навыки планирования, организации и структурирования кода, что помогает при проектировании программ и архитектуры приложений. Основное внимание уделено языку программирования C++, где были усвоены основы объектно-ориентированного программирования (ООП), что позволило эффективно использовать его возможности при написании игровой логики.

Полученный опыт и знания могут быть применены в будущих проектах и задачах разработки программного обеспечения. В будущем планируется улучшение текущего функционала. Планируется добавить новые возможности, такие как ведение нотации, возможность играть с искусственным интеллектом.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Объектно-ориентированное программирование в С++, – Роберт Лафоре, 2019

[2] Язык программирования С++, – Бьерн Страуструп, 1985

[3] Qt 5.10 Профессиональное программирование на С++, – Макс

Шлее, 2018

[4] Qt Documentation [Электронный ресурс]. -Электронные данные.  
-Режим доступа: <https://doc.qt.io/> - Дата доступа: 20.10.2023

[5] Chess.com [Электронный ресурс]. -Электронные данные. – Режим доступа: <https://www.chess.com/> - Дата доступа: 20.09.2023

**ПРИЛОЖЕНИЕ А**

(обязательное)

Диаграмма классов

**ПРИЛОЖЕНИЕ Б**

(обязательное)

Схема алгоритма обработки нажатия на фигуру

**ПРИЛОЖЕНИЕ В**

(обязательное)

Схема алгоритма обработки нажатия на ячейку поля

**ПРИЛОЖЕНИЕ Г**

(обязательное)

Листинг кода

Файл main.cpp:

#include "mainwindow.h"

#include <QApplication>

int

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

{

QApplication a (argc, argv);

MainWindow menu;

menu.show ();

return a.exec ();

}

Файл bishop.h:

#ifndef BISHOP\_H

#define BISHOP\_H

#include "chesspiece.h"

class Bishop : public ChessPiece

{

public:

Bishop (QString team, QGraphicsItem \*parent = 0);

void setImage ();

void moves ();

};

#endif // BISHOP\_H

Файл bishop.cpp:

#include "bishop.h"

#include "game.h"

#include <QDebug>

extern Game \*game;

Bishop::Bishop (QString team, QGraphicsItem \*parent) : ChessPiece (team, parent) { setImage (); }

void

Bishop::setImage ()

{

if (side == "WHITE")

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/whitebishop.png"));

else

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/blackbishop.png"));

}

void

Bishop::moves ()

{

location.clear ();

int row = this->getCurrentBox ()->rowLoc;

int col = this->getCurrentBox ()->colLoc;

QString team = this->getSide ();

// For upper Left

for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

{

break;

}

}

}

// For upper right

for (int i = row - 1, j = col + 1; i >= 0 && j <= 7; i--, j++)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

{

break;

}

}

}

// For downward right

for (int i = row + 1, j = col + 1; i <= 7 && j <= 7; i++, j++)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

{

break;

}

}

}

// For downward left

for (int i = row + 1, j = col - 1; i <= 7 && j >= 0; i++, j--)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

{

break;

}

}

}

}

Файл button.h:

#ifndef BUTTON\_H

#define BUTTON\_H

#include <QGraphicsRectItem>

#include <QGraphicsSceneMouseEvent>

class Button : public QObject, public QGraphicsRectItem

{

Q\_OBJECT

public:

Button (QString name, QGraphicsItem \*parent = NULL);

// public methods

void setFont (QFont const textFont);

// events

void mousePressEvent (QGraphicsSceneMouseEvent \*event);

void hoverEnterEvent (QGraphicsSceneHoverEvent \*event);

void hoverLeaveEvent (QGraphicsSceneHoverEvent \*event);

signals:

void clicked ();

private:

QGraphicsTextItem \*text;

};

#endif // BUTTON\_H

Файл button.cpp:

#include "button.h"

#include "qfont.h"

#include <QBrush>

#include <QGraphicsTextItem>

Button::Button (QString name, QGraphicsItem \*parent) : QGraphicsRectItem (parent)

{

// draw the rect

setRect (0, 0, 150, 50);

QBrush brush;

brush.setStyle (Qt::SolidPattern);

brush.setColor (QColor::fromRgb (qRgb (61, 61, 61)));

setBrush (brush);

// draw Text

text = new QGraphicsTextItem (name, this);

int xPos = rect ().width () / 2 - text->boundingRect ().width () / 2 - 15;

int yPos = rect ().height () / 2 - text->boundingRect ().height () / 2;

text->setPos (xPos, yPos);

text->setDefaultTextColor (QColor::fromRgb (qRgb (247, 247, 247)));

// Allow responding to hover

setAcceptHoverEvents (true);

}

void

Button::setFont (QFont const textFont)

{

text->setFont (textFont);

}

void

Button::mousePressEvent (QGraphicsSceneMouseEvent \*event)

{

if (event)

emit clicked ();

}

void

Button::hoverEnterEvent (QGraphicsSceneHoverEvent \*event)

{

// change color

if (event)

{

QBrush brush;

brush.setStyle (Qt::SolidPattern);

brush.setColor (QColor::fromRgb (qRgb (127, 127, 127)));

setBrush (brush);

}

}

void

Button::hoverLeaveEvent (QGraphicsSceneHoverEvent \*event)

{

// change color

if (event)

{

QBrush brush;

brush.setStyle (Qt::SolidPattern);

brush.setColor (QColor::fromRgb (qRgb (61, 61, 61)));

setBrush (brush);

}

}

Файл chessboard.h:

#ifndef CHESSBOARD\_H

#define CHESSBOARD\_H

#include "chesspiece.h"

#include <QGraphicsRectItem>

class ChessBoard

{

public:

ChessBoard ();

// drawing public function

void drawBoxes (int x, int y);

void addChessPiece ();

void setUpWhite ();

void setUpBlack ();

void reset ();

private:

QList<ChessPiece \*> white;

QList<ChessPiece \*> black;

};

#endif // CHESSBOARD\_H

Файл chessboard.cpp:

#include "bishop.h"

#include "chessboard.h"

#include "chessbox.h"

#include "game.h"

#include "king.h"

#include "knight.h"

#include "pawn.h"

#include "queen.h"

#include "rook.h"

extern Game \*game;

ChessBoard::ChessBoard ()

{

setUpBlack ();

setUpWhite ();

}

void

ChessBoard::drawBoxes (int x, int y)

{

int SHIFT = 100;

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

ChessBox \*box = new ChessBox ();

game->collection[i][j] = box;

box->rowLoc = i;

box->colLoc = j;

box->setPos (x + SHIFT \* j, y + SHIFT \* i);

if ((i + j) % 2 == 0)

box->setOriginalColor (QColor::fromRgb (qRgb (247, 247, 247))); // white

else

box->setOriginalColor (QColor::fromRgb (qRgb (118, 118, 118))); // black

game->addToScene (box);

}

}

}

void

ChessBoard::addChessPiece ()

{

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

ChessBox \*box = game->collection[i][j];

if (i < 2)

{

static int k;

box->placePiece (black[k]);

game->alivePiece.append (black[k]);

game->addToScene (black[k++]);

}

if (i > 5)

{

static int h;

box->placePiece (white[h]);

game->alivePiece.append (white[h]);

game->addToScene (white[h++]);

}

}

}

}

void

ChessBoard::setUpWhite ()

{

ChessPiece \*piece;

for (int i = 0; i < 8; i++)

{

piece = new Pawn ("WHITE");

white.append (piece);

}

piece = new Rook ("WHITE");

white.append (piece);

piece = new Knight ("WHITE");

white.append (piece);

piece = new Bishop ("WHITE");

white.append (piece);

piece = new Queen ("WHITE");

white.append (piece);

piece = new King ("WHITE");

white.append (piece);

piece = new Bishop ("WHITE");

white.append (piece);

piece = new Knight ("WHITE");

white.append (piece);

piece = new Rook ("WHITE");

white.append (piece);

}

void

ChessBoard::setUpBlack ()

{

ChessPiece \*piece;

piece = new Rook ("BLACK");

black.append (piece);

piece = new Knight ("BLACK");

black.append (piece);

piece = new Bishop ("BLACK");

black.append (piece);

piece = new Queen ("BLACK");

black.append (piece);

piece = new King ("BLACK");

black.append (piece);

piece = new Bishop ("BLACK");

black.append (piece);

piece = new Knight ("BLACK");

black.append (piece);

piece = new Rook ("BLACK");

black.append (piece);

for (int i = 0; i < 8; i++)

{

piece = new Pawn ("BLACK");

black.append (piece);

}

}

void

ChessBoard::reset ()

{

int k = 0;

int h = 0;

for (int i = 0; i < 8; i++)

{

for (int j = 0; j < 8; j++)

{

ChessBox \*box = game->collection[i][j];

box->setHasChessPiece (false);

box->setChessPieceColor ("NONE");

box->currentPiece = NULL;

if (i < 2)

{

box->placePiece (black[k]);

black[k]->setIsPlaced (true);

black[k]->firstMove = true;

game->alivePiece.append (black[k++]);

}

if (i > 5)

{

box->placePiece (white[h]);

white[h]->setIsPlaced (true);

white[h]->firstMove = true;

game->alivePiece.append (white[h++]);

}

}

}

}

Файл chessbox.h :

#ifndef CHESSBOX\_H

#define CHESSBOX\_H

#include "chesspiece.h"

#include <QBrush>

#include <QGraphicsRectItem>

#include <QGraphicsSceneMouseEvent>

class ChessPiece;

class ChessBox : public QGraphicsRectItem

{

public:

// Constructor

ChessBox (QGraphicsItem \*parent = 0);

~ChessBox ();

// public member function

void mousePressEvent (QGraphicsSceneMouseEvent \*event);

void setColor (QColor color);

void placePiece (ChessPiece \*piece);

void resetOriginalColor ();

void setOriginalColor (QColor value);

bool getHasChessPiece ();

void setHasChessPiece (bool value, ChessPiece \*piece = 0);

void checkForCheck ();

QString getChessPieceColor ();

void setChessPieceColor (QString value);

int rowLoc;

int colLoc;

ChessPiece \*currentPiece;

private:

QBrush brush;

QColor originalColor;

bool hasChessPiece;

QString chessPieceColor;

};

#endif // CHESSBOX\_H

Файл chessbox.cpp:

#include "chessbox.h"

#include "game.h"

#include "king.h"

#include <QDebug>

extern Game \*game;

ChessBox::ChessBox (QGraphicsItem \*parent) : QGraphicsRectItem (parent)

{

// making the Square CHess Box

setRect (0, 0, 100, 100);

brush.setStyle (Qt::SolidPattern);

setZValue (-1);

setHasChessPiece (false);

setChessPieceColor ("NONE");

currentPiece = NULL;

}

ChessBox::~ChessBox () { delete this; }

void

ChessBox::mousePressEvent (QGraphicsSceneMouseEvent \*event)

{

// Deselecting counter part of ChessPiece

if (currentPiece == game->pieceToMove && currentPiece)

{

currentPiece->mousePressEvent (event);

return;

}

// if selected

if (game->pieceToMove)

{

// if same team

if (this->getChessPieceColor () == game->pieceToMove->getSide ())

return;

// removing the eaten piece

QList<ChessBox \*> movLoc = game->pieceToMove->moveLocation ();

// TO make sure the selected box is in move zone

int check = 0;

for (size\_t i = 0, n = movLoc.size (); i < n; i++)

{

if (movLoc[i] == this)

{

check++;

}

}

// if not prsent return

if (check == 0)

return;

// change the color back to normal

game->pieceToMove->decolor ();

// make the first move false applicable for pawn only

game->pieceToMove->firstMove = false;

// this is to eat or consume the enemy present inn the movable region

if (this->getHasChessPiece ())

{

this->currentPiece->setIsPlaced (false);

this->currentPiece->setCurrentBox (NULL);

game->placeInDeadPlace (this->currentPiece);

}

// changing the new stat and resetting the previous left region

game->pieceToMove->getCurrentBox ()->setHasChessPiece (false);

game->pieceToMove->getCurrentBox ()->currentPiece = NULL;

game->pieceToMove->getCurrentBox ()->resetOriginalColor ();

placePiece (game->pieceToMove);

game->pieceToMove = NULL;

// changing turn

game->changeTurn ();

checkForCheck ();

}

// Selecting couterpart of the chessPiece

else if (this->getHasChessPiece ())

{

this->currentPiece->mousePressEvent (event);

}

}

void

ChessBox::setColor (QColor color)

{

brush.setColor (color);

setBrush (color);

}

void

ChessBox::placePiece (ChessPiece \*piece)

{

piece->setPos (x () + 50 - piece->pixmap ().width () / 2,

y () + 50 - piece->pixmap ().width () / 2);

piece->setCurrentBox (this);

setHasChessPiece (true, piece);

currentPiece = piece;

}

void

ChessBox::resetOriginalColor ()

{

setColor (originalColor);

}

void

ChessBox::setOriginalColor (QColor value)

{

originalColor = value;

setColor (originalColor);

}

bool

ChessBox::getHasChessPiece ()

{

return hasChessPiece;

}

void

ChessBox::setHasChessPiece (bool value, ChessPiece \*piece)

{

hasChessPiece = value;

if (value)

setChessPieceColor (piece->getSide ());

else

setChessPieceColor ("NONE");

}

void

ChessBox::checkForCheck ()

{

int c = 0;

QList<ChessPiece \*> pList = game->alivePiece;

for (size\_t i = 0, n = pList.size (); i < n; i++)

{

King \*p = dynamic\_cast<King \*> (pList[i]);

if (p)

{

continue;

}

pList[i]->moves ();

pList[i]->decolor ();

QList<ChessBox \*> bList = pList[i]->moveLocation ();

for (size\_t j = 0, n = bList.size (); j < n; j++)

{

King \*p = dynamic\_cast<King \*> (bList[j]->currentPiece);

if (p)

{

if (p->getSide () == pList[i]->getSide ())

continue;

bList[j]->setColor (Qt::darkMagenta);

// if(pList[i]->getCurrentBox()->getChessPieceColor() == "WHITE")

// emit game->whiteWon();

// else

// emit game->blackWon();

// game->gameOver();

// pList[i]->getCurrentBox()->setColor(Qt::darkYellow);

if (!game->check->isVisible ())

game->check->setVisible (true);

else

{

bList[j]->resetOriginalColor ();

pList[i]->getCurrentBox ()->resetOriginalColor ();

// game->gameOver();

}

c++;

}

}

}

if (!c)

{

game->check->setVisible (false);

for (size\_t i = 0, n = pList.size (); i < n; i++)

pList[i]->getCurrentBox ()->resetOriginalColor ();

}

}

QString

ChessBox::getChessPieceColor ()

{

return chessPieceColor;

}

void

ChessBox::setChessPieceColor (QString value)

{

chessPieceColor = value;

}

Файл chesspiece.h:

#ifndef CHESSPIECE\_H

#define CHESSPIECE\_H

#include "chessbox.h"

#include <QGraphicsPixmapItem>

#include <QGraphicsSceneMouseEvent>

class ChessBox;

class ChessPiece : public QGraphicsPixmapItem

{

public:

ChessPiece (QString team = "", QGraphicsItem \*parent = 0);

void mousePressEvent (QGraphicsSceneMouseEvent \*event);

void setCurrentBox (ChessBox \*box);

ChessBox \*getCurrentBox ();

QString getSide ();

void setSide (QString value);

virtual void setImage () = 0;

bool getIsPlaced ();

void setIsPlaced (bool value);

QList<ChessBox \*> moveLocation ();

virtual void moves () = 0;

void decolor ();

bool firstMove;

bool boxSetting (ChessBox \*box);

protected:

ChessBox \*currentBox;

QString side;

bool isPlaced;

QList<ChessBox \*> location;

};

#endif // CHESSPIECE\_H

Файл chesspiece.cpp:

#include "chesspiece.h"

#include "game.h"

#include "king.h"

#include <QDebug>

extern Game \*game;

ChessPiece::ChessPiece (QString team, QGraphicsItem \*parent) : QGraphicsPixmapItem (parent)

{

side = team;

isPlaced = true;

firstMove = true;

}

void

ChessPiece::mousePressEvent (QGraphicsSceneMouseEvent \*event)

{

// Deselect

if (this == game->pieceToMove)

{

game->pieceToMove->getCurrentBox ()->resetOriginalColor ();

game->pieceToMove->decolor ();

game->pieceToMove = NULL;

return;

}

// if it is already consumed or not the respective color's turn

if ((!getIsPlaced ()) || ((game->getTurn () != this->getSide ()) && (!game->pieceToMove)))

return;

// selecting

if (!game->pieceToMove)

{

game->pieceToMove = this;

game->pieceToMove->getCurrentBox ()->setColor (Qt::darkGreen);

game->pieceToMove->moves ();

}

// Consuming counterPart of the CHessBox

else if (this->getSide () != game->pieceToMove->getSide ())

{

this->getCurrentBox ()->mousePressEvent (event);

}

}

void

ChessPiece::setCurrentBox (ChessBox \*box)

{

currentBox = box;

}

ChessBox \*

ChessPiece::getCurrentBox ()

{

return currentBox;

}

QString

ChessPiece::getSide ()

{

return side;

}

void

ChessPiece::setSide (QString value)

{

side = value;

}

bool

ChessPiece::getIsPlaced ()

{

return isPlaced;

}

void

ChessPiece::setIsPlaced (bool value)

{

isPlaced = value;

}

QList<ChessBox \*>

ChessPiece::moveLocation ()

{

return location;

}

void

ChessPiece::decolor ()

{

for (size\_t i = 0, n = location.size (); i < n; i++)

{

location[i]->resetOriginalColor ();

}

}

bool

ChessPiece::boxSetting (ChessBox \*box)

{

if (box->getHasChessPiece ())

{

King \*q = dynamic\_cast<King \*> (location.last ()->currentPiece);

if (q)

{

box->setColor (Qt::darkMagenta);

}

else

box->setColor (Qt::yellow);

return true;

}

else

location.last ()->setColor (Qt::green);

return false;

}

Файл game.h:

#ifndef GAME\_H

#define GAME\_H

#include "chesspiece.h"

#include <QGraphicsScene>

#include <QGraphicsView>

#include <chessboard.h>

class Game : public QGraphicsView

{

Q\_OBJECT

public:

// Constructors

Game (QWidget \*parent = 0);

// public Methods

void drawDeadHolder (int x, int y, QColor color);

void drawChessBoard ();

void displayDeadWhite ();

void displayDeadBlack ();

void placeInDeadPlace (ChessPiece \*piece);

void drawPawnMenu ();

// Scene Related

void addToScene (QGraphicsItem \*item);

void removeFromScene (QGraphicsItem \*item);

// getters/setters

ChessPiece \*pieceToMove;

QString getTurn ();

void setTurn (QString value);

void changeTurn ();

ChessBox \*collection[8][8];

QGraphicsTextItem \*check;

QList<ChessPiece \*> alivePiece;

void displayMainMenu ();

void gameOver ();

void removeAll ();

signals:

void GameOver ();

void whiteWon ();

void blackWon ();

public slots:

void start ();

private:

QGraphicsScene \*gameScene;

QList<ChessPiece \*> whiteDead;

QList<ChessPiece \*> blackDead;

QGraphicsRectItem \*deadHolder;

QGraphicsRectItem \*pawnMenu;

QString turn;

ChessBoard \*chess;

QList<QGraphicsItem \*> listG;

QGraphicsTextItem \*turnDisplay;

QGraphicsTextItem \*choosePawnText;

};

#endif // GAME\_H

Файл game.cpp:

#include "button.h"

#include "game.h"

#include "king.h"

#include <QDebug>

#include <QPixmap>

Game::Game (QWidget \*parent) : QGraphicsView (parent)

{

// Making the Scene

gameScene = new QGraphicsScene ();

gameScene->setSceneRect (0, 0, 1400, 900);

// Making the view

setFixedSize (1400, 900);

setHorizontalScrollBarPolicy (Qt::ScrollBarAlwaysOff);

setVerticalScrollBarPolicy (Qt::ScrollBarAlwaysOff);

setScene (gameScene);

QBrush brush;

brush.setStyle (Qt::SolidPattern);

brush.setColor (QColor::fromRgb (qRgb (61, 61, 61)));

setBackgroundBrush (brush);

pieceToMove = NULL;

// display turn

turnDisplay = new QGraphicsTextItem ();

turnDisplay->setPos (width () / 2 - 100, 10);

turnDisplay->setZValue (1);

turnDisplay->setDefaultTextColor (QColor::fromRgb (qRgb (247, 247, 247)));

QFont fontTurn;

fontTurn.setFamily (QString::fromUtf8 ("Courier"));

fontTurn.setPointSize (18);

fontTurn.setBold (true);

fontTurn.setKerning (true);

turnDisplay->setFont (fontTurn);

turnDisplay->setPlainText ("Turn : WHITE");

// display Check

check = new QGraphicsTextItem ();

check->setPos (width () / 2 - 100, 860);

check->setZValue (4);

check->setDefaultTextColor (Qt::red);

QFont fontCheck;

fontCheck.setFamily (QString::fromUtf8 ("Courier"));

fontCheck.setPointSize (18);

fontCheck.setBold (true);

fontCheck.setKerning (true);

check->setFont (fontCheck);

check->setPlainText ("CHECK!");

check->setVisible (false);

setTurn ("WHITE");

}

void

Game::drawChessBoard ()

{

chess = new ChessBoard ();

drawDeadHolder (0, 0, QColor::fromRgb (qRgb (247, 247, 247)));

drawDeadHolder (1100, 0, QColor::fromRgb (qRgb (247, 247, 247)));

chess->drawBoxes (width () / 2 - 400, 50);

}

void

Game::displayDeadWhite ()

{

int SHIFT = 50;

int j = 0;

int k = 0;

for (size\_t i = 0, n = whiteDead.size (); i < n; i++)

{

if (j == 4)

{

k++;

j = 0;

}

whiteDead[i]->setPos (40 + SHIFT \* j++, 100 + SHIFT \* 2 \* k);

}

}

void

Game::displayDeadBlack ()

{

int SHIFT = 50;

int j = 0;

int k = 0;

for (size\_t i = 0, n = blackDead.size (); i < n; i++)

{

if (j == 4)

{

k++;

j = 0;

}

blackDead[i]->setPos (1140 + SHIFT \* j++, 100 + SHIFT \* 2 \* k);

}

}

void

Game::placeInDeadPlace (ChessPiece \*piece)

{

if (piece->getSide () == "WHITE")

{

whiteDead.append (piece);

King \*g = dynamic\_cast<King \*> (piece);

if (g)

{

check->setPlainText ("Black Won");

emit blackWon ();

gameOver ();

}

displayDeadWhite ();

}

else

{

blackDead.append (piece);

King \*g = dynamic\_cast<King \*> (piece);

if (g)

{

check->setPlainText ("White Won");

emit whiteWon ();

gameOver ();

}

displayDeadBlack ();

}

alivePiece.removeAll (piece);

}

void

Game::addToScene (QGraphicsItem \*item)

{

gameScene->addItem (item);

}

void

Game::removeFromScene (QGraphicsItem \*item)

{

gameScene->removeItem (item);

}

QString

Game::getTurn ()

{

return turn;

}

void

Game::setTurn (QString value)

{

turn = value;

}

void

Game::changeTurn ()

{

if (getTurn () == "WHITE")

setTurn ("BLACK");

else

setTurn ("WHITE");

turnDisplay->setPlainText ("Turn : " + getTurn ());

}

void

Game::start ()

{

drawChessBoard ();

for (size\_t i = 0, n = listG.size (); i < n; i++)

removeFromScene (listG[i]);

addToScene (turnDisplay);

QGraphicsTextItem \*whitePiece = new QGraphicsTextItem ();

whitePiece->setPos (70, 10);

whitePiece->setZValue (1);

whitePiece->setDefaultTextColor (QColor::fromRgb (qRgb (61, 61, 61)));

QFont fontWhitePiece;

fontWhitePiece.setFamily (QString::fromUtf8 ("Courier"));

fontWhitePiece.setPointSize (20);

fontWhitePiece.setBold (true);

fontWhitePiece.setKerning (true);

whitePiece->setFont (fontWhitePiece);

whitePiece->setPlainText ("WHITE PIECE");

addToScene (whitePiece);

QGraphicsTextItem \*blackPiece = new QGraphicsTextItem ();

blackPiece->setPos (1170, 10);

blackPiece->setZValue (1);

blackPiece->setDefaultTextColor (QColor::fromRgb (qRgb (61, 61, 61)));

QFont fontBlackPiece;

fontBlackPiece.setFamily (QString::fromUtf8 ("Courier"));

fontBlackPiece.setPointSize (20);

fontBlackPiece.setBold (true);

fontBlackPiece.setKerning (true);

blackPiece->setFont (fontBlackPiece);

blackPiece->setPlainText ("BLACK PIECE");

addToScene (blackPiece);

addToScene (check);

chess->addChessPiece ();

// drawPawnMenu();

}

void

Game::drawDeadHolder (int x, int y, QColor color)

{

deadHolder = new QGraphicsRectItem (x, y, 300, 900);

QBrush brush;

brush.setStyle (Qt::SolidPattern);

brush.setColor (color);

deadHolder->setBrush (brush);

addToScene (deadHolder);

}

void

Game::drawPawnMenu ()

{

pawnMenu = new QGraphicsRectItem (300, 400, 800, 200);

QBrush brush;

brush.setStyle (Qt::SolidPattern);

brush.setColor (QColor::fromRgb (qRgb (247, 247, 247)));

pawnMenu->setBrush (brush);

addToScene (pawnMenu);

choosePawnText = new QGraphicsTextItem ();

turnDisplay->setPos (450, 420);

turnDisplay->setZValue (1);

turnDisplay->setDefaultTextColor (QColor::fromRgb (qRgb (61, 61, 61)));

QFont font;

font.setFamily (QString::fromUtf8 ("Courier"));

font.setPointSize (18);

font.setBold (true);

font.setKerning (true);

turnDisplay->setFont (font);

turnDisplay->setPlainText ("Choose a piece instead of a pawn");

Button \*bishopButton = new Button ("bishop");

Button \*knightButton = new Button ("knight");

Button \*queenButton = new Button ("queen");

Button \*rookButton = new Button ("rook");

int pxPos = 350;

int pyPos = 500;

bishopButton->setPos (pxPos, pyPos);

knightButton->setPos (pxPos + 175, pyPos);

queenButton->setPos (pxPos + 350, pyPos);

rookButton->setPos (pxPos + 525, pyPos);

font.setPointSize (12);

bishopButton->setFont (font);

knightButton->setFont (font);

queenButton->setFont (font);

rookButton->setFont (font);

connect (bishopButton, SIGNAL (clicked ()), this, SLOT (start ()));

addToScene (bishopButton);

addToScene (knightButton);

addToScene (queenButton);

addToScene (rookButton);

listG.append (bishopButton);

listG.append (knightButton);

listG.append (queenButton);

listG.append (rookButton);

}

void

Game::gameOver ()

{

// removeAll();

// setTurn("WHITE");

// alivePiece.clear();

// chess->reset();

this->close ();

emit GameOver ();

}

void

Game::removeAll ()

{

QList<QGraphicsItem \*> itemsList = gameScene->items ();

for (size\_t i = 0, n = itemsList.size (); i < n; i++)

{

if (itemsList[i] != check)

removeFromScene (itemsList[i]);

}

}

Файл king.h:

#ifndef KING\_H

#define KING\_H

#include "chesspiece.h"

class King : public ChessPiece

{

public:

King (QString team, QGraphicsItem \*parent = 0);

void setImage ();

void findUnSafeLocation ();

void moves ();

};

#endif // KING\_H

Файл king.cpp:

#include "game.h"

#include "king.h"

#include "pawn.h"

extern Game \*game;

King::King (QString team, QGraphicsItem \*parent) : ChessPiece (team, parent) { setImage (); }

void

King::setImage ()

{

if (side == "WHITE")

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/whiteking.png"));

else

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/blackking.png"));

}

void

King::moves ()

{

location.clear ();

int row = this->getCurrentBox ()->rowLoc;

int col = this->getCurrentBox ()->colLoc;

QString team = this->getSide ();

if (col > 0 && row > 0 && !(game->collection[row - 1][col - 1]->getChessPieceColor () == team))

{ // up left

location.append (game->collection[row - 1][col - 1]);

game->collection[row - 1][col - 1]->setColor (Qt::green);

if (location.last ()->getHasChessPiece ())

{

location.last ()->setColor (Qt::yellow);

}

}

if (col < 7 && row > 0 && !(game->collection[row - 1][col + 1]->getChessPieceColor () == team))

{ // up right

location.append (game->collection[row - 1][col + 1]);

game->collection[row - 1][col + 1]->setColor (Qt::green);

if (location.last ()->getHasChessPiece ())

{

location.last ()->setColor (Qt::yellow);

}

}

if (row > 0 && !(game->collection[row - 1][col]->getChessPieceColor () == team))

{ // up

location.append (game->collection[row - 1][col]);

game->collection[row - 1][col]->setColor (Qt::green);

if (location.last ()->getHasChessPiece ())

{

location.last ()->setColor (Qt::yellow);

}

}

if (row < 7 && !(game->collection[row + 1][col]->getChessPieceColor () == team))

{ // down

location.append (game->collection[row + 1][col]);

game->collection[row + 1][col]->setColor (Qt::green);

if (location.last ()->getHasChessPiece ())

{

location.last ()->setColor (Qt::yellow);

}

}

if (col > 0 && !(game->collection[row][col - 1]->getChessPieceColor () == team))

{ // left

location.append (game->collection[row][col - 1]);

game->collection[row][col - 1]->setColor (Qt::green);

if (location.last ()->getHasChessPiece ())

{

location.last ()->setColor (Qt::yellow);

}

}

if (col < 7 && !(game->collection[row][col + 1]->getChessPieceColor () == team))

{ // right

location.append (game->collection[row][col + 1]);

game->collection[row][col + 1]->setColor (Qt::green);

if (location.last ()->getHasChessPiece ())

{

location.last ()->setColor (Qt::yellow);

}

}

if (col > 0 && row < 7 && !(game->collection[row + 1][col - 1]->getChessPieceColor () == team))

{ // down left

location.append (game->collection[row + 1][col - 1]);

game->collection[row + 1][col - 1]->setColor (Qt::green);

if (location.last ()->getHasChessPiece ())

{

location.last ()->setColor (Qt::yellow);

}

}

if (col < 7 && row < 7 && !(game->collection[row + 1][col + 1]->getChessPieceColor () == team))

{ // down right

location.append (game->collection[row + 1][col + 1]);

game->collection[row + 1][col + 1]->setColor (Qt::green);

if (location.last ()->getHasChessPiece ())

{

location.last ()->setColor (Qt::yellow);

}

}

// findUnSafeLocation();

}

void

King::findUnSafeLocation ()

{

QList<ChessPiece \*> pList = game->alivePiece;

for (size\_t i = 0, n = pList.size (); i < n; i++)

{

if (pList[i]->getSide () != this->getSide ())

{

QList<ChessBox \*> bList = pList[i]->moveLocation ();

for (size\_t j = 0, n = bList.size (); j < n; j++)

{

Pawn \*c = dynamic\_cast<Pawn \*> (pList[i]);

if (c)

continue;

for (size\_t k = 0, n = location.size (); k < n; k++)

{

if (bList[j] == location[k])

{

location[k]->setColor (Qt::blue);

}

}

}

}

}

}

Файл knight.h:

#ifndef KNIGHT\_H

#define KNIGHT\_H

#include "chesspiece.h"

class Knight : public ChessPiece

{

public:

Knight (QString team, QGraphicsItem \*parent = 0);

void setImage ();

void moves ();

};

#endif // KNIGHT\_H

Файл knight.cpp:

#include "game.h"

#include "knight.h"

extern Game \*game;

Knight::Knight (QString team, QGraphicsItem \*parent) : ChessPiece (team, parent) { setImage (); }

void

Knight::setImage ()

{

if (side == "WHITE")

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/whiteknight.png"));

else

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/blackknight.png"));

}

void

Knight::moves ()

{

int row = this->getCurrentBox ()->rowLoc;

int col = this->getCurrentBox ()->colLoc;

int i, j;

QString team = this->getSide ();

// There are total 8 places where a night can move

// 1st up up left

i = row - 2;

j = col - 1;

if (i >= 0 && j >= 0 && (game->collection[i][j]->getChessPieceColor () != team))

{

location.append (game->collection[i][j]);

if (location.last ()->getHasChessPiece ())

location.last ()->setColor (Qt::yellow);

else

location.last ()->setColor (Qt::green);

}

// 2nd up up right

i = row - 2;

j = col + 1;

if (i >= 0 && j <= 7 && (game->collection[i][j]->getChessPieceColor () != team))

{

location.append (game->collection[i][j]);

if (location.last ()->getHasChessPiece ())

location.last ()->setColor (Qt::yellow);

else

location.last ()->setColor (Qt::green);

}

// 3rd down down left

i = row + 2;

j = col - 1;

if (i <= 7 && j >= 0 && (game->collection[i][j]->getChessPieceColor () != team))

{

location.append (game->collection[i][j]);

if (location.last ()->getHasChessPiece ())

location.last ()->setColor (Qt::yellow);

else

location.last ()->setColor (Qt::green);

}

// 4th down down right

i = row + 2;

j = col + 1;

if (i <= 7 && j <= 7 && (game->collection[i][j]->getChessPieceColor () != team))

{

location.append (game->collection[i][j]);

if (location.last ()->getHasChessPiece ())

location.last ()->setColor (Qt::yellow);

else

location.last ()->setColor (Qt::green);

}

// 5th left left up

i = row - 1;

j = col - 2;

if (i >= 0 && j >= 0 && (game->collection[i][j]->getChessPieceColor () != team))

{

location.append (game->collection[i][j]);

if (location.last ()->getHasChessPiece ())

location.last ()->setColor (Qt::yellow);

else

location.last ()->setColor (Qt::green);

}

// 6th left left down

i = row + 1;

j = col - 2;

if (i <= 7 && j >= 0 && (game->collection[i][j]->getChessPieceColor () != team))

{

location.append (game->collection[i][j]);

if (location.last ()->getHasChessPiece ())

location.last ()->setColor (Qt::yellow);

else

location.last ()->setColor (Qt::green);

}

// 7th right right up

i = row - 1;

j = col + 2;

if (i >= 0 && j <= 7 && (game->collection[i][j]->getChessPieceColor () != team))

{

location.append (game->collection[i][j]);

if (location.last ()->getHasChessPiece ())

location.last ()->setColor (Qt::yellow);

else

location.last ()->setColor (Qt::green);

}

// 8th right right down

i = row + 1;

j = col + 2;

if (i <= 7 && j <= 7 && (game->collection[i][j]->getChessPieceColor () != team))

{

location.append (game->collection[i][j]);

if (location.last ()->getHasChessPiece ())

location.last ()->setColor (Qt::yellow);

else

location.last ()->setColor (Qt::green);

}

}

Файл mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

QT\_BEGIN\_NAMESPACE

namespace Ui

{

class MainWindow;

}

QT\_END\_NAMESPACE

class MainWindow : public QMainWindow

{

Q\_OBJECT

public:

MainWindow (QWidget \*parent = nullptr);

~MainWindow ();

private slots:

void on\_buttonEXIT\_clicked ();

void on\_buttonPVP\_clicked ();

public slots:

void whiteWON ();

void blackWON ();

private:

Ui::MainWindow \*ui;

};

#endif // MAINWINDOW\_H

Файл mainwindow.cpp

#include "game.h"

#include "mainwindow.h"

#include "ui\_mainwindow.h"

Game \*game;

MainWindow::MainWindow (QWidget \*parent) : QMainWindow (parent), ui (new Ui::MainWindow)

{

ui->setupUi (this);

}

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

void

MainWindow::on\_buttonEXIT\_clicked ()

{

this->close ();

}

void

MainWindow::whiteWON ()

{

this->setStyleSheet (QString::fromUtf8 ("background-image:url(E:/kursovaya3sem/myProject/media/"

"whitewon.png); background-position: center; "));

}

void

MainWindow::blackWON ()

{

this->setStyleSheet (QString::fromUtf8 ("background-image:url(E:/kursovaya3sem/myProject/media/"

"blackwon.png); background-position: center; "));

}

void

MainWindow::on\_buttonPVP\_clicked ()

{

if (game != nullptr)

{

game->close ();

}

this->ui->buttonPVP->hide ();

this->hide ();

game = new Game ();

game->show ();

game->start ();

QObject::connect (game, SIGNAL (GameOver ()), this, SLOT (show ()));

QObject::connect (game, SIGNAL (whiteWon ()), this, SLOT (whiteWON ()));

QObject::connect (game, SIGNAL (blackWon ()), this, SLOT (blackWON ()));

}

Файл pawn.h:

#ifndef PAWN\_H

#define PAWN\_H

#include "chesspiece.h"

class Pawn : public ChessPiece

{

public:

Pawn (QString team, QGraphicsItem \*parent = 0);

void setImage ();

void moves ();

private:

};

#endif // PAWN\_H

Файл pawn.cpp:

#include "game.h"

#include "king.h"

#include "pawn.h"

#include <QDebug>

#include <typeinfo>

extern Game \*game;

Pawn::Pawn (QString team, QGraphicsItem \*parent) : ChessPiece (team, parent)

{

setImage ();

firstMove = true;

}

void

Pawn::setImage ()

{

if (side == "WHITE")

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/whitepawn.png"));

else

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/blackpawn.png"));

}

void

Pawn::moves ()

{

location.clear ();

int row = this->getCurrentBox ()->rowLoc;

int col = this->getCurrentBox ()->colLoc;

if (this->getSide () == "WHITE")

{

if (col > 0 && row > 0

&& game->collection[row - 1][col - 1]->getChessPieceColor () == "BLACK")

{

location.append (game->collection[row - 1][col - 1]);

boxSetting (location.last ());

}

if (col < 7 && row > 0

&& game->collection[row - 1][col + 1]->getChessPieceColor () == "BLACK")

{

location.append (game->collection[row - 1][col + 1]);

boxSetting (location.last ());

}

if (row > 0 && (!game->collection[row - 1][col]->getHasChessPiece ()))

{

location.append (game->collection[row - 1][col]);

boxSetting (location.last ());

if (firstMove && !game->collection[row - 2][col]->getHasChessPiece ())

{

location.append (game->collection[row - 2][col]);

boxSetting (location.last ());

}

}

}

else

{

if (col > 0 && row < 7

&& game->collection[row + 1][col - 1]->getChessPieceColor () == "WHITE")

{ // left

location.append (game->collection[row + 1][col - 1]);

boxSetting (location.last ());

}

if (col < 7 && row < 7

&& game->collection[row + 1][col + 1]->getChessPieceColor () == "WHITE")

{ // right

location.append (game->collection[row + 1][col + 1]);

boxSetting (location.last ());

}

if (row < 7 && (!game->collection[row + 1][col]->getHasChessPiece ()))

{

location.append (game->collection[row + 1][col]);

boxSetting (location.last ());

if (firstMove && !game->collection[row + 2][col]->getHasChessPiece ())

{

location.append (game->collection[row + 2][col]);

boxSetting (location.last ());

}

}

}

}

Файл queen.h:

#ifndef QUEEN\_H

#define QUEEN\_H

#include "chesspiece.h"

class Queen : public ChessPiece

{

public:

Queen (QString team, QGraphicsItem \*parent = 0);

void setImage ();

void moves ();

};

#endif // QUEEN\_H

Файл queen.cpp:

#include "game.h"

#include "queen.h"

extern Game \*game;

Queen::Queen (QString team, QGraphicsItem \*parent) : ChessPiece (team, parent) { setImage (); }

void

Queen::setImage ()

{

if (side == "WHITE")

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/whitequeen.png"));

else

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/blackqueen.png"));

}

void

Queen::moves ()

{

location.clear ();

int row = this->getCurrentBox ()->rowLoc;

int col = this->getCurrentBox ()->colLoc;

QString team = this->getSide ();

// For up

for (int i = row - 1, j = col; i >= 0; i--)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

break;

}

}

// For Down

for (int i = row + 1, j = col; i <= 7; i++)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

{

break;

}

}

}

// For left

for (int i = row, j = col - 1; j >= 0; j--)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

break;

}

}

// For Right

for (int i = row, j = col + 1; j <= 7; j++)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

break;

}

}

// For upper Left

for (int i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

{

break;

}

}

}

// For upper right

for (int i = row - 1, j = col + 1; i >= 0 && j <= 7; i--, j++)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

{

break;

}

}

}

// For downward right

for (int i = row + 1, j = col + 1; i <= 7 && j <= 7; i++, j++)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

{

break;

}

}

}

// For downward left

for (int i = row + 1, j = col - 1; i <= 7 && j >= 0; i++, j--)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

{

break;

}

}

}

}

Файл rook.h:

#ifndef ROOK\_H

#define ROOK\_H

#include "chesspiece.h"

class Rook : public ChessPiece

{

public:

Rook (QString team, QGraphicsItem \*parent = 0);

void setImage ();

void moves ();

};

#endif // ROOK\_H

Файл rook.cpp:

#include "game.h"

#include "rook.h"

extern Game \*game;

Rook::Rook (QString team, QGraphicsItem \*parent) : ChessPiece (team, parent) { setImage (); }

void

Rook::setImage ()

{

if (side == "WHITE")

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/whiterook.png"));

else

setPixmap (QPixmap ("E:/kursovaya3sem/myProject/media/blackrook.png"));

}

void

Rook::moves ()

{

location.clear ();

int row = this->getCurrentBox ()->rowLoc;

int col = this->getCurrentBox ()->colLoc;

QString team = this->getSide ();

// For up

for (int i = row - 1, j = col; i >= 0; i--)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

break;

}

}

// For Down

for (int i = row + 1, j = col; i <= 7; i++)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

{

break;

}

}

}

// For left

for (int i = row, j = col - 1; j >= 0; j--)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

break;

}

}

// For Right

for (int i = row, j = col + 1; j <= 7; j++)

{

if (game->collection[i][j]->getChessPieceColor () == team)

{

break;

}

else

{

location.append (game->collection[i][j]);

if (boxSetting (location.last ()))

break;

}

}

}

**ПРИЛОЖЕНИЕ Д**

(обязательное)

Ведомость документов