МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

Работа допущена к защите

_____Руководитель

«____»_____20___г.

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование на языке C++»

на тему: «Разработка игры "Шахматы"»

Студент _____ Бессонов М.П.

Шифр 191009

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 92ПГ

Руководитель _____ Захарова О.В.

Оценка: «_____» Дата _____

Орел 2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

УТВЕРЖДАЮ:

_____Зав. кафедрой

«___»_____20__г.

ЗАДАНИЕ
на курсовую работу

по дисциплине «Объектно-ориентированное программирование на языке С++»

Студент Бессонов М.П. Шифр 191009

Институт приборостроения, автоматизации и информационных технологий

Направление подготовки 09.03.04 «Программная инженерия»

Группа 92ПГ

1 Тема курсовой работы

«Разработка игры "Шахматы"»

2 Срок сдачи студентом законченной работы «10» июня 2020

3 Исходные данные

Условие задания, алгоритмы ходов шахматных фигур, правила шахматной игры

4 Содержание курсовой работы

Постановка задачи и разработка требований

Обоснование выбора методов решения и структур данных

Проектирование программы

Особенности программной реализации

Описание пользовательского интерфейса

5 Отчетный материал курсовой работы

Пояснительная записка курсовой работы; презентация

Руководитель _____ Захарова О.В.

Задание принял к исполнению: «20» февраля 2020

Подпись студента_____

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире существует множество методов, применяемых на различных стадиях жизненного цикла программного обеспечения (ПО). Совокупность таких методов получила название методология разработки ПО. Каждая методология характеризуется своими концепциями и основными принципами, которые позволяют четко ее классифицировать [3].

В этой работе мы рассмотрим одну из самых известных методологий программирования - объектно-ориентированное программирование (ООП).

Цель нашей работы: опираясь на принципы ООП и основные этапы разработки ПО, разработать программу "Шахматы" для двух игроков.

Эта тема является актуальной, так как объектно-ориентированный подход к программированию как к моделированию информационных объектов решает на новом уровне основную задачу структурного программирования: структурирование информации с точки зрения управляемости [5].

Для достижения цели нам необходимо решить соответствующие задачи:

1) выбор метода решения поставленной задачи;
2) выбор структур данных;
3) разработка структуры программы и алгоритмов;
4) реализация программы на языке C++;
5) описание пользовательского интерфейса.

1 ПОСТАНОВКА ЗАДАЧИ И РАЗРАБОТКА ТРЕБОВАНИЙ

В программе разработать родительский класс "Фигура" и шесть дочерних классов (с соблюдением инкапсуляции). Кроме того, разработать еще два класса, один из которых отвечает за размещение фигур на доске, а второй за реализацию игры. Программа должна иметь удобный графический интерфейс.

Основной функционал программы:

1) перемещение фигуры при вводе соответствующих координат;
2) смена цвета с одного на другой после осуществления хода;
3) отображение всей информации на шахматном поле, изменяющемся после каждого хода;
4) отображение сообщений о корректности хода;
5) отображение сообщения о завершении игры.

Основной функционал реализовывать с соблюдением принципа полиморфизма. При разработке программы продемонстрировать понимание основных принципов объектно-ориентированного программирования (инкапсуляция, наследование, полиморфизм) [4].

Разработаем требования, в соответствии с которыми мы будем реализовывать нашу программу. В нашей работе мы рассмотрим пользовательские требования (описывают цели/задачи пользователей системы, которые должны достигаться/выполняться пользователями при помощи создаваемой программной системы) и функциональные требования (определяют функциональность (поведение) программной системы) [8].

К пользовательским требованиям нашей программы можно отнести следующие:

- возможность совершать ход конкретной фигурой при выборе соответствующих координат в соответствии с правилами шахмат;
- возможность видеть изменения шахматной доски после хода;
- возможность видеть сообщения о корректности своего хода и о

завершении игры.

К функциональным требованиям нашей программы можно отнести следующие:

- программа должна изменять положение фигур на графической доске при выборе координат перемещения;
- программа должна отображать доску по соответствующему действию пользователя;
- программа должна представлять текущую информацию об игре и всех объектах в ней в виде графического шахматного поля;
- программа должна сменять игрока (менять цвет) после хода текущего;
- программа должна выдавать сообщения о корректности хода;
- программа должна выдать сообщение о победе какого-либо игрока или о патовой ситуации.

В группу функциональных требований относят и системные требования [1]. Эти характеристики могут описывать требования как к аппаратному обеспечению (тип и частота процессора, объём оперативной памяти, объём жесткого диска), так и к программному окружению (операционная система, наличие установленных системных компонентов и сервисов и т. п.).

Если рассматривать системные требования нашей программы, то можно выделить следующие:

- Тип системы: 64-разрядная операционная система;
- Процессор: Intel Core i5;
- Оперативная память: 8 ГБ и более;
- Операционная система: Windows 7;
- Жесткий диск: 200 ГБ и более.

2 ОБОСНОВАНИЕ ВЫБОРА МЕТОДОВ РЕШЕНИЯ И СТРУКТУР ДАННЫХ

Для реализации задачи можно воспользоваться несколькими методами решения:

1) Вводить информацию о ходах в числовой форме, преобразовывать ее в начальные и конечные координаты и изменять положение фигур на доске;

2) Вводить информацию о ходах в стандартной форме (буква-число), преобразовывать ее в начальные и конечные координаты и изменять положение фигур на доске.

Учитывая тот факт, что обычному пользователю будет быстрее и удобнее работать с числовой информацией, следует, что более рациональным методом решения поставленной задачи является метод, при котором ход игрока будет вводится одним числом. Данный способ удобен тем, что позволяет не использовать множество условий или отдельный массив для буквенных обозначений, а позволяет использовать стандартные операции взятия целой части и остатка.

Для реализации полиморфизма и определения того, какой фигуре следует ходить, мы будем использовать виртуальные функции - функции, которые переопределяются для каждого дочернего класса. Такое решение позволяет структурировать код таким образом, чтобы новые дочерние классы автоматически работали со старым кодом без необходимости внесения изменений со стороны программиста [7].

Для взаимодействия пользователя с программой необходимо разработать удобный интерфейс. Существует несколько способов его реализации:

1) консольный интерфейс;
2) графический интерфейс.

Консольный интерфейс потребляет небольшое количество ресурсов и позволяет выводить большое количество текстовой информации.

Графический интерфейс удобен тем, что представляет всю информацию на дисплее в виде графических изображений.

Учитывая тот факт, что пользователь больше привык работать с графическим интерфейсом, так как это позволяет визуально оценить происходящее в программе и облегчает понимание и освоение программы неподготовленным пользователем. Исходя из всего вышеперечисленного, был сделан выбор в пользу графического интерфейса.

Для большей удобности ввод координат будет осуществляться в отдельных диалоговых окнах [6], выводящихся последовательно. Кроме того, для удобства пользователя в них будут выводиться сообщения о корректности хода, соответствии цвета, угрозе короля и завершении игры.

Структура данных предназначена для того, чтобы хранить и обрабатывать множество однотипных данных в вычислительной технике.

Для реализации данной задачи можно использовать разные виды структур данных:

1) класс;
2) структура;
3) массив.

Учитывая тот факт, что наши фигуры должны будут размещаться на доске 8*8, мы будем использовать двумерный массив той же размерности.

Так как класс, в отличие от структуры, позволяет объединить все функции и свойства в одно целое и предотвратить их изменение вне класса, придерживаясь принципа инкапсуляции, мы поместим наш двумерный массив в классе, прописав в конструкторе начальное положение каждой фигуры на доске.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММЫ

Программа состоит из пяти модулей.

На рисунке 1 представлена модульная схема программы.

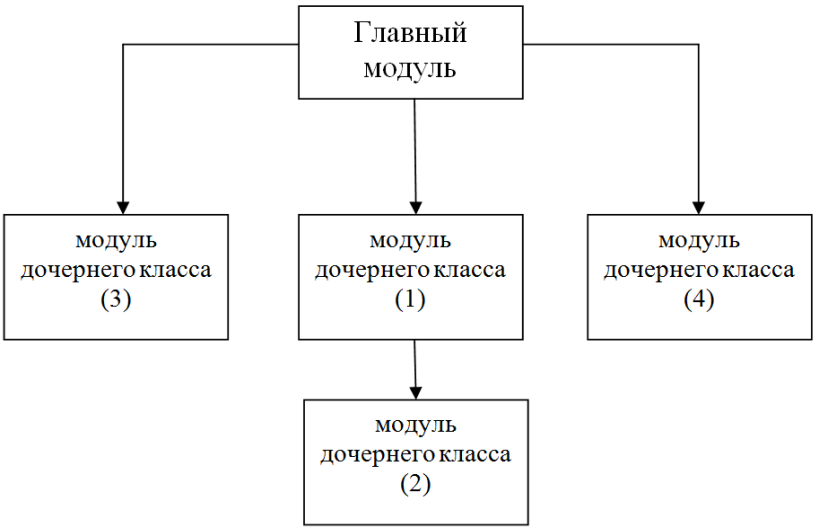

Рисунок 1 – Схема модульной структуры программы

Главный модуль - модуль главного окна, содержащий методы для обработки нажатий на кнопки, для изменения фигур на доске и для вывода дополнительных сообщений.

Модуль дочернего класса (1), который включает один класс, хранящий в себе базовый класс "Фигура", содержащий в себе виртуальные методы, необходимые для переопределения текущей фигуры и ее возможного хода. Кроме того, в этом классе находятся методы определения цвета фигуры и проверки ее законного хода.

Модуль дочернего класса (2), который содержит в себе шесть классов фигур ("Пешка", "Конь", "Слон", "Ладья", "Королева", "Король"). У них почти одинаковое внутреннее содержание. Все они переопределяют функцию, которая определяет название фигуры. Также у каждого класса разное тело виртуальной функции "возможный ход" в зависимости от того, как ходит та или иная фигура и какие проверки для этого нужны.

Модуль дочернего класса (3) содержит в себе один класс - игровую доску ("Доска"). Этот класс содержит указатель на двумерный массив ("Фигура* доска[8][8]"), конструктор, который размещает в этом массиве объекты классов фигур, а также функцию "Обработчик", которая возвращает название и цвет фигуры.

Модуль дочернего класса (4) также содержит в себе один класс - "Шахматы", в котором реализуется сама игра. Он содержит открытый конструктор, в который с самого начала передан цвет "W", так как ход начинают белые. Также в нем содержится функция "СледующийХод" в которой происходит изменение положения фигуры после соответствующих проверок, функция "АльтернативныйХод", которая изменяет текущего игрока (меняет цвет), и логическая функция "КонецИгры", которая, в зависимости от выполнения условий, выводит сообщение "Checkmate! W Wins", "Checkmate! B Wins" или "Stalemate". Закрытым свойством в этом классе является "ЦветИгрока". Диаграмма классов [9] нашей программы представлена на рисунке 2.

Опишем основные алгоритмы, которые используются в нашей программе.

Прежде всего - это алгоритм поиска короля. Он представляет собой вложенный цикл с несколькими условиями, который предназначен для того, чтобы, пройдя по всему нашему двумерному массиву 8*8 найти координаты короля текущего игрока и записать их в координаты "строка короля" и "столбец короля". Схему данного алгоритма вы можете видеть на рисунке 3.

Второй используемый алгоритм - это алгоритм "анализа короля под боем". Он реализован с помощью вложенного цикла с несколькими условиями, в котором мы проходим по всей доске и находим хотя бы одну фигуру, которая своим следующим ходом может побить короля. После этого в независимую переменную записывается результат этого поиска: "true", если есть хотя бы одна такая фигура, и "false", если ее нет. Схему данного алгоритма вы можете видеть на рисунке 4.

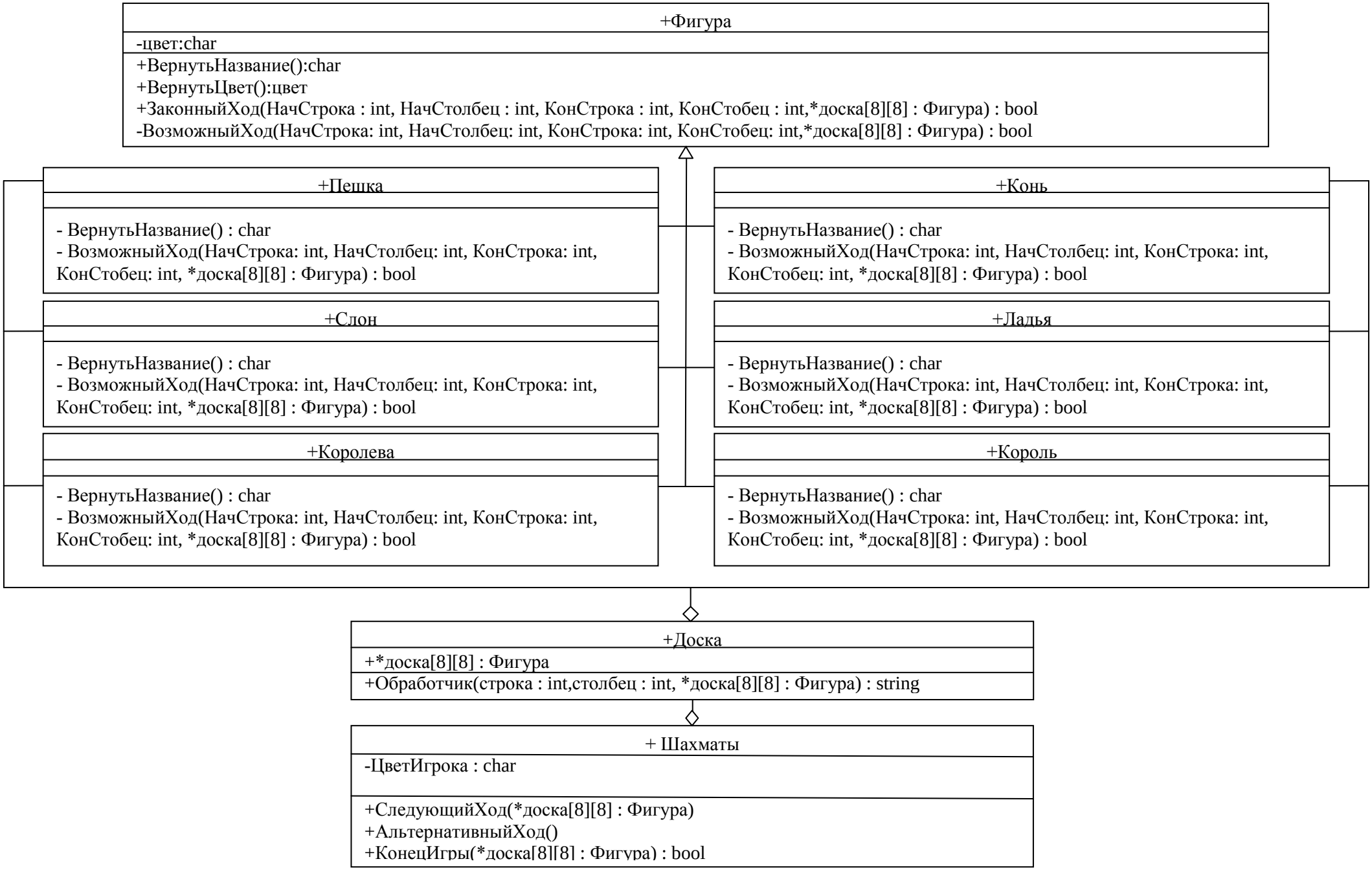

Рисунок 2 – Диаграмма классов

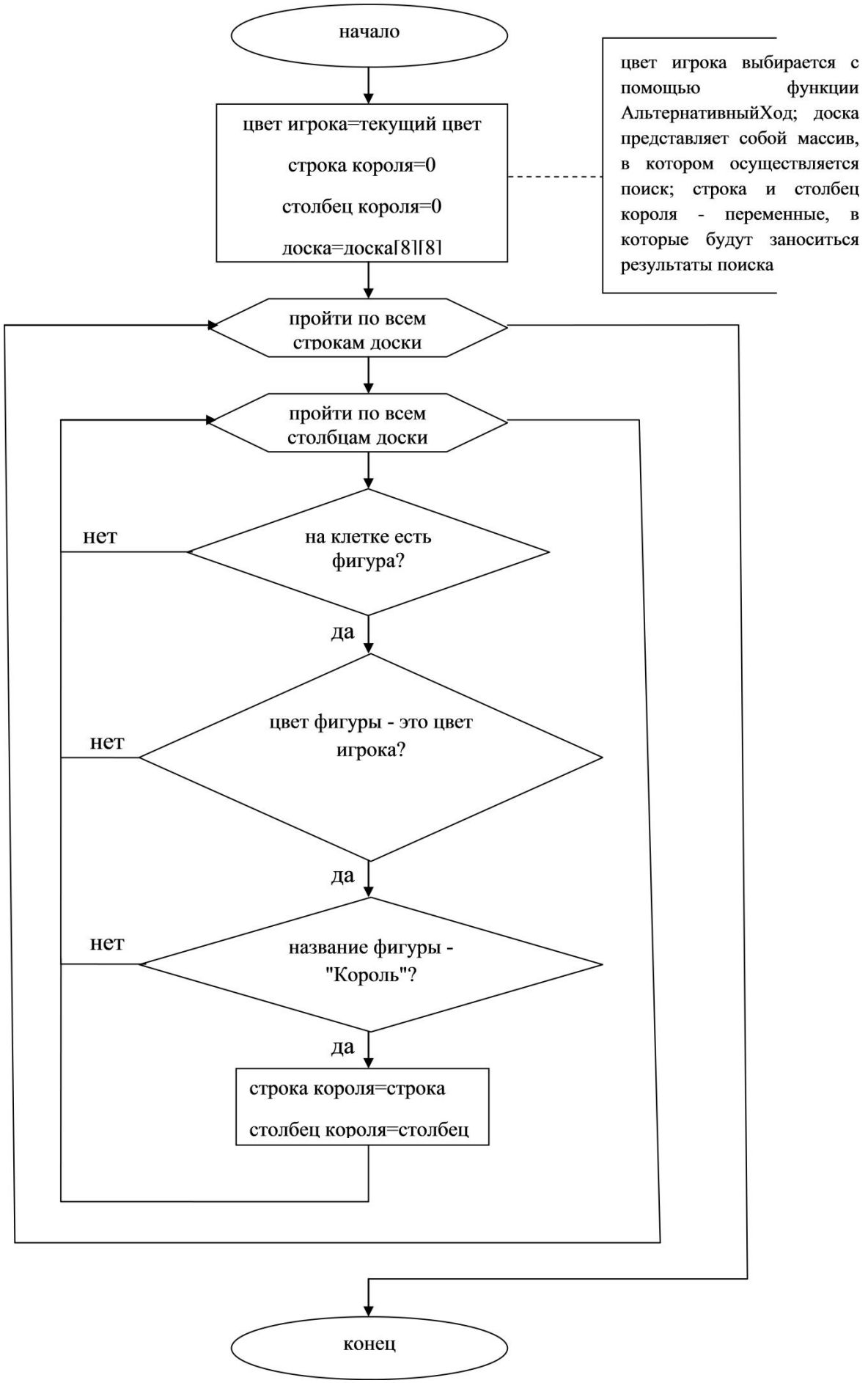

Рисунок 3 – Схема алгоритма поиска короля

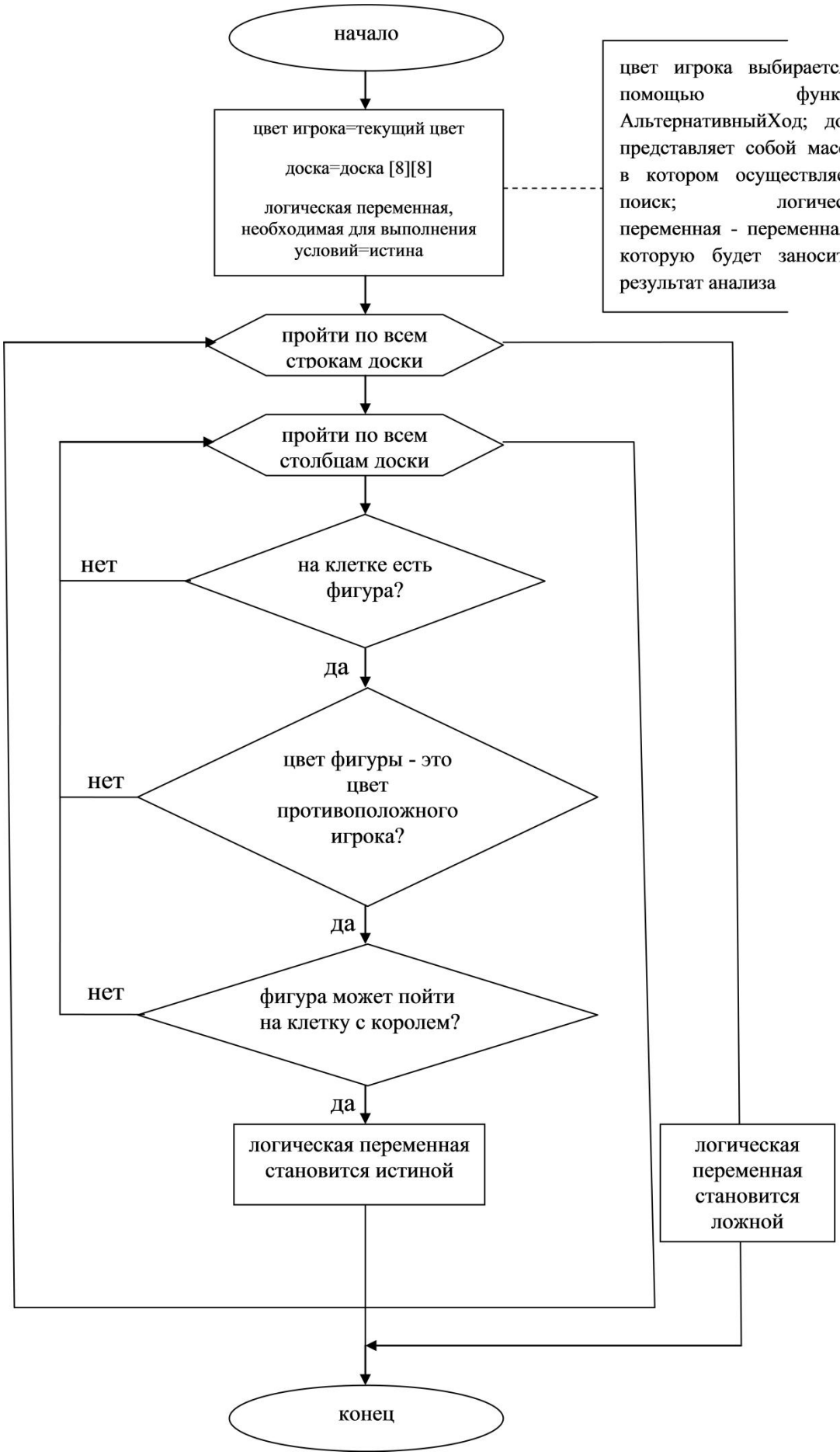

Рисунок 4 – Схема алгоритма анализа короля под боем

4 ОСОБЕННОСТИ ПРОГРАММНОЙ РЕАЛИЗАЦИИ

Главной структурой программы является класс.

Класс "CAPiece" (приложение Б) является базовым абстрактным классом с двумя чисто виртуальными функциями. Он нужен только для того, чтобы мы могли работать с производными классами фигур. Этот класс содержит одно закрытое поле "mcColor" типа "char", которое хранит в себе цвет текущей фигуры. Также этот класс содержит следующие функции:

1. Публичный метод "GetColor" типа "char", который возвращает текущий цвет ("mcColor").

2. Публичный чисто виртуальный метод "GetPiece" типа "char", который возвращает название фигуры.

3. Публичный метод "IsLegalMove" типа "bool", который проводит первичную проверку законности хода (если клетка, на которую пойдет фигура, пустая, или на ней стоит фигура противника, то ход законный).

4. Закрытый чисто виртуальный публичный метод "AreSquaresLegal" типа "bool", который вызывается из функции "IsLegalMove" и, соответственно, вызывает один из наших производных классов в зависимости от переданных координат.

Наследниками этого классы являются шесть классов фигур (приложение В). У всех у них одинаковая структура: они все содержат две виртуальные функции базового класса ("GetPiece" и "AreSquaresLegal"):

1. Закрытый метод "GetPiece" переопределяет название фигуры, которое содержится в нашем массиве и используется для вывода фигур на экран.

2. Закрытый метод "AreSquaresLegal" проверяет может ли данная фигура совершить ход, который ввел пользователь.

Класс "CBoard" (приложение Г) имеет одно открытое поле "CAPiece* mqpaaBoard[8][8]", которое является указателем на наш двумерный массив и используется для изменения положения фигур. Также этот класс содержит

одну открытую "obr" функцию типа "string". Эта функция возвращает цвет и название фигуры в одной строке (например, "WP" - белая пешка) или значение "0", если клетка пуста.

Класс "CChess" (приложение Д) содержит в себе одно закрытое поле - "mcPlayerTurn" (цвет текущего игрока) типа "char", которое нужно для определения текущего хода и различных проверок. Также этот класс содержит следующие публичные методы:

1. "GetNextMove" типа "void", который преобразует вводимые в диалоговых окнах координата в начальное и конечное положение и, после проверок на корректность ввода координат, цвета и хода, изменяет положение этого объекта в нашем двумерном массиве.

2. "AlternateTurn" типа "void", который изменяет нашу переменную цвета "mcPlayerTurn" на противоположный цвет с помощью тернарной операции.

3. "IsGameOver" типа "bool", который проверяет есть ли хотя бы один ход у текущего игрока, при котором его король не будет находиться под боем. Если его нет, запускается алгоритм поиска короля и анализа короля под боем. Если следующим ходом короля могут побить и нет ходов, чтобы его защитить, вызывается функция "AlternateTurn" и выводится сообщение о победе. Если у текущего игрока ходов нет, но его король "в безопасности", выводится сообщение о пате.

Для реализации графического интерфейса мы будем использовать графический редактор, работающий с формами, wxWidgets [10]. Графический интерфейс (приложение А) реализован с помощью класса "_1grFrame" в заголовочном файле " _1grFrame.h", в котором размещены все формы, использующиеся в нашей программе.

Взаимодействие с пользовательским интерфейсом представлено в виде двух кнопок "Print" и "Next". За их реализацию отвечают функции "OnButton3Click" и "OnButton5Click".

Метод void "OnButton3Click" класса "_1grFrame" позволяет обработать нажатие кнопки "Print", которая, использует функции "obr", "isblack" типа "bool" (проверяет клетку и возвращает "true", если клетка черная). Затем, исходя из результата этих двух функций, в другом указателе на двумерный массив создаются новые объекты изображений фигур.

Метод void "OnButton5Click" класса "_1grFrame" позволяет обработать нажатие кнопки "Next". Этот метод содержит в себе три функции ("GetNextMove", "AlternateTurn" и "IsGameOver"), из класса "CChess". Для их вызова мы в самом начале главного модуля создали объект класса "CChess" и объект класса "CBoard".

5 ОПИСАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА

Опишем основной интерфейс нашего приложения. На экране приложения расположен заголовок окна со стандартными кнопками "закрыть", "свернуть", "развернуть". Также экран содержит главное меню с кнопкой "File", в которой содержится выход из приложения. Экран содержит основную рабочую область. В нашем случае это шахматная доска и две кнопки (Рисунок 5).

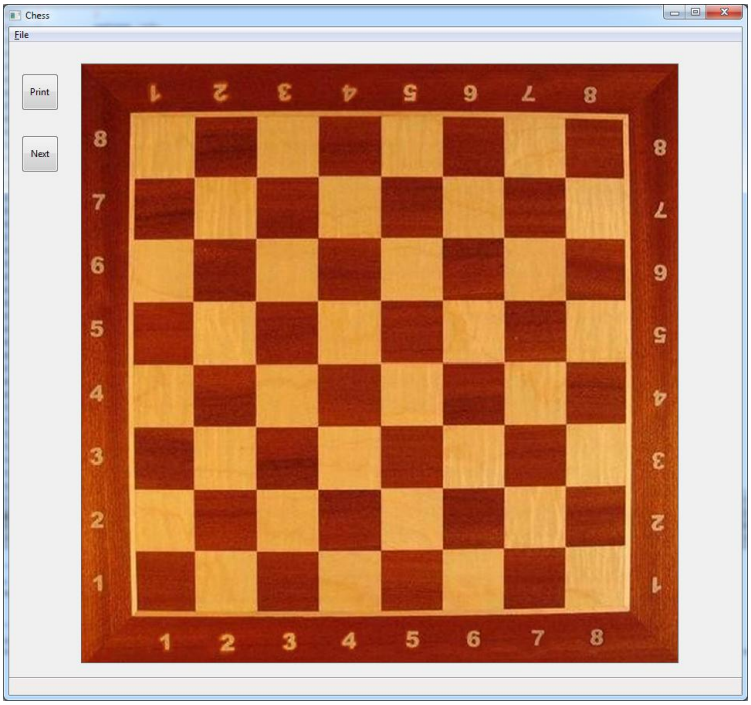

Рисунок 5 – Шахматы

Основное взаимодействие с программой происходит с помощью двух кнопок - "Print" и "Next". Их выбор осуществляется нажатием левой кнопки мыши.

После нажатия кнопки "Next" на экране появляются последовательно два поля ввода (Рисунок 6) [6].

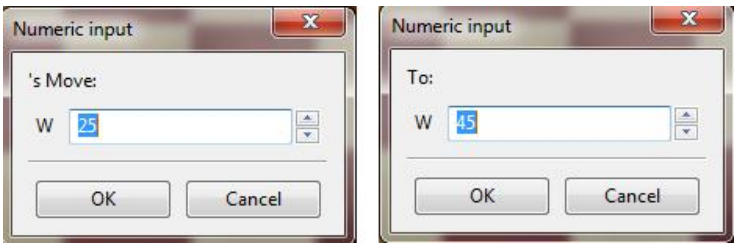

Рисунок 6 – Диалоговые окна для ввода

Поле ввода представляет собой элемент управления, который дает пользователю возможность ввести информацию. Поля в нашей программе используют только числовую информацию (ввод другого вида информации невозможен), для того чтобы ввести координаты фигуры. Ввод информации в них может осуществляться как вручную, так и с помощью полосы прокрутки. Координаты нужно вводить одним числом, сначала строка, потом столбец (Рисунок 6). В любом другом случае программа посчитает данные некорректными.

После ввода координат запускается процесс их обработки, который может выдать один из нескольких представленных результатов:

– если все координаты верны - программы выдаст на экран три последовательных диалоговых сообщения "Color right", "Move legal", "No alarm". После этого соответствующая фигура изменит свое положение на доске. Чтобы увидеть эти изменения, нужно нажать на кнопку "Print" (Рисунок 7);

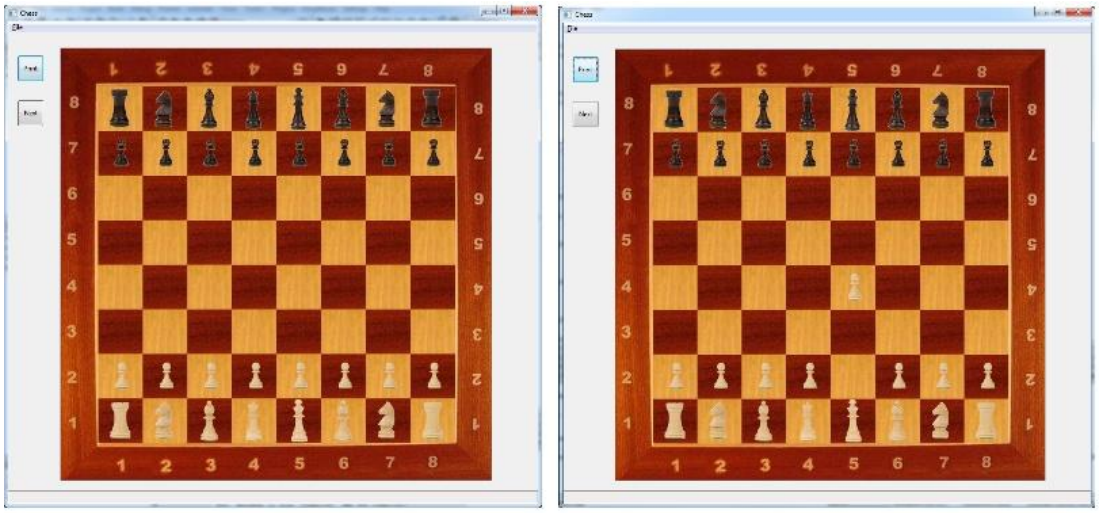

Рисунок 7 – Работа кнопки "Print"

– если цвет фигуры верный, но ход совершен неправильно, то на экране появятся только два сообщения "Color right" и "Invalid move!". После этого снова появятся два поля ввода для повторного введения данных;

– если цвет неверный или введенные данные некорректны, то появляется сообщение "Invalid move!" и, после его закрытия, два поля ввода;

– если цвет верный, ход правильный, но король текущей стороны находится под ударом (состояние "Шах" в шахматах), то выведутся сообщения "Color right", "Move legal", "Alarm!", "Invalid move!". После снова появятся два поля ввода.

По окончании игры на экран выведется одно из трех возможных сообщений "Checkmate! W Wins", "Checkmate! B Wins", "Stalemate" (Рисунок 8). Далее приложение закроется.

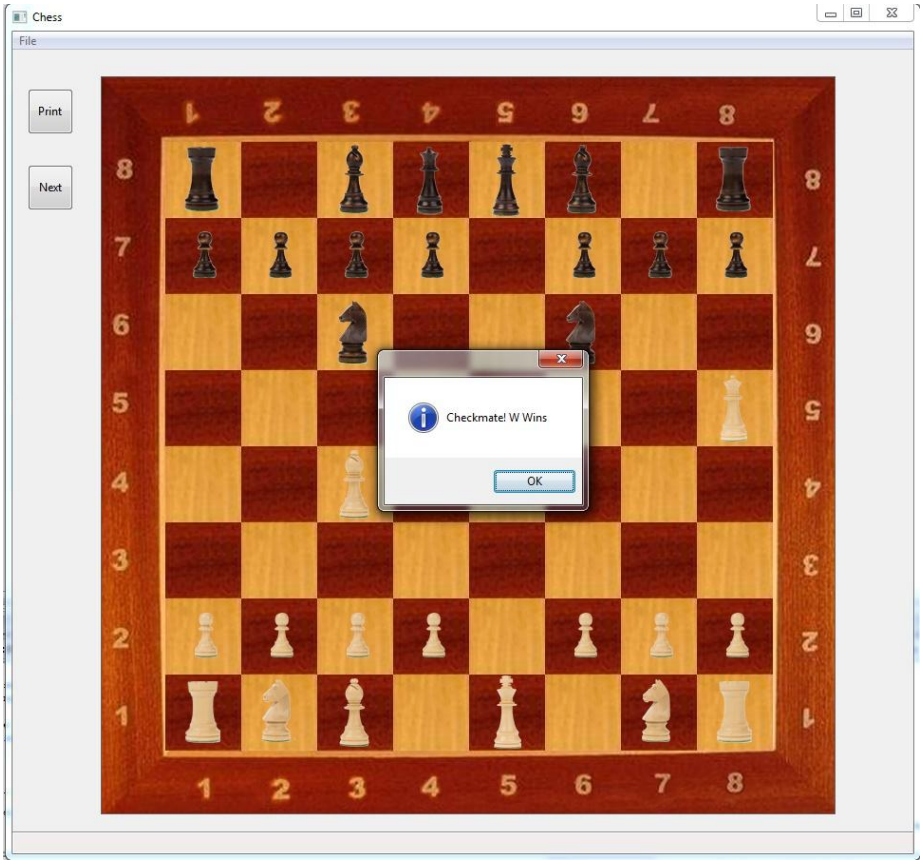

Рисунок 8 – Сообщение о победе

Сам игровой процесс представляет собой обычную шахматную партию с некоторыми исключениями: нет рокировки, взятия на проходе и замены пешки по достижении края доски. Но реализована возможность первого хода пешки на две клетки [2].

ЗАКЛЮЧЕНИЕ

В курсовой работе была рассмотрена одна из самых известных методологий программирования - объектно-ориентированное программирование. Также были рассмотрены ее основные принципы и возможности их реализации.

Была создана и описана программа "Шахматы" для двух игроков на языке C++.

Мы разработали основные требования к программе, спроектировали ее и рассмотрели особенности реализации. Мы выяснили, что у каждого способа есть свои достоинства и недостатки. Одни повышают наглядность и понятность программы, но снижают ее эффективность по памяти и времени. Другие позволяют эффективнее работать с памятью, но усложняют саму структуру программы. Кроме того, был описан и разработан пользовательский интерфейс.

СПИСОК ЛИТЕРАТУРЫ

1. Виды требований. Все для аналитиков [Электронный ресурс]. - Режим доступа: http://foranalysts.blogspot.com/2011/08/blog-post_17.html (дата обращения: 5.04.2020).

2. Как играть в шахматы | Правила + 7 шагов для начинающих [Электронный ресурс]. - Режим доступа: https://www.chess.com/ru/kak-igrat-v-shakhmaty (дата обращения: 5.04.2020).

3. Методология программирования [Электронный ресурс]. - Режим доступа:https://ru.wikipedia.org/wiki/Методология_программирования (дата обращения: 5.04.2020).

4. Объектно-ориентированное программирование [Электронный ресурс]. - Режим доступа: https://prog-cpp.ru/oop/ (дата обращения: 5.04.2020).

5. Объектно-ориентированный подход к программированию Library [Электронный ресурс]. - Режим доступа: https://cyberpedia.su/22x19c1.html (дата обращения: 5.04.2020).

6. Руководство по wxwidgets программирование [Электронный ресурс]. - Режим доступа: https://docplayer.ru/54713725-Rukovodstvo-po-wxwidgets.html (дата обращения: 5.04.2020).

7. Урок №163. Виртуальные функции и Полиморфизм [Электронный ресурс]. - Режим доступа: https://ravesli.com/urok-163-virtualnye-funktsii-i-polimorfizm/ (дата обращения: 5.04.2020).

8. Функциональные и нефункциональные требования (Functional and Non-functional Requirements) [Электронный ресурс]. - Режим доступа: https://studfile.net/preview/2152457/page:4/ (дата обращения: 5.04.2020).

9. UML-диаграммы классов программирование [Электронный ресурс]. - Режим доступа: https://prog-cpp.ru/uml-classes/ (дата обращения: 5.04.2020).

10. wxWidgets: Cross-Platform GUI Library [Электронный ресурс]. - Режим доступа: https://www.wxwidgets.org/ (дата обращения: 5.04.2020).

ПРИЛОЖЕНИЕ А

(обязательное)

ЛИСТИНГ ФАЙЛА "_1grMain.cpp"

```
#include "wx_pch.h"
#include "_1grMain.h"
#include <wx/msgdlg.h>
#include <string>
#include <iostream>
#include <cmath>
#include "wx/numdlg.h"
#include <stdio.h>
#include <stdlib.h>
#include "CAPiece.h"
#include "CFigure.h"
#include "CBoard.h"
#include "CChess.h"

//(*InternalHeaders(_1grFrame)
#include <wx/bitmap.h>
#include <wx/image.h>
#include <wx/intl.h>
#include <wx/string.h>
//*)

//helper functions
enum wxbuildinfoformat {
   short_f, long_f };
int i=0, j=0, smX=160, smY=655, s=0;
char g='W';

wxStaticBitmap* cell[8][8];
wxStaticBitmap* grafboard;
wxString wxbuildinfo(wxbuildinfoformat format)
{
   wxString wxbuild(wxVERSION_STRING);

   if (format == long_f )
   {
#if defined(__WXMSW__)
      wxbuild << _T("-Windows");
#elif defined(__UNIX__)
      wxbuild << _T("-Linux");
#endif

#if wxUSE_UNICODE
      wxbuild << _T("-Unicode build");
#else
      wxbuild << _T("-ANSI build");
#endif // wxUSE_UNICODE
```

```
    }

    return wxbuild;
}

//(*IdInit(_1grFrame)
const long _1grFrame::ID_BUTTON1 = wxNewId();
const long _1grFrame::ID_STATICBITMAP2 = wxNewId();
const long _1grFrame::ID_STATICBITMAP1 = wxNewId();
const long _1grFrame::ID_BUTTON3 = wxNewId();
const long _1grFrame::ID_STATICTEXT1 = wxNewId();
const long _1grFrame::ID_TEXTCTRL1 = wxNewId();
const long _1grFrame::ID_TEXTCTRL2 = wxNewId();
const long _1grFrame::ID_STATICTEXT2 = wxNewId();
const long _1grFrame::ID_BUTTON5 = wxNewId();
const long _1grFrame::ID_PANEL1 = wxNewId();
const long _1grFrame::idMenuQuit = wxNewId();
const long _1grFrame::ID_STATUSBAR1 = wxNewId();
//*)

BEGIN_EVENT_TABLE(_1grFrame,wxFrame)
    //(*EventTable(_1grFrame)
    //*)
END_EVENT_TABLE()

_1grFrame::_1grFrame(wxWindow* parent,wxWindowID id)
{
    //(*Initialize(_1grFrame)
    wxMenu* Menu1;
    wxMenuBar* MenuBar1;
    wxMenuItem* MenuItem1;

    Create(parent, wxID_ANY, wxEmptyString, wxDefaultPosition, wxDefaultSize,
wxDEFAULT_FRAME_STYLE, _T("wxID_ANY"));
    SetClientSize(wxSize(949,864));
    Panel1 = new wxPanel(this, ID_PANEL1, wxPoint(144,80), wxSize(960,888),
wxTAB_TRAVERSAL, _T("ID_PANEL1"));
    Button1 = new wxButton(Panel1, ID_BUTTON1, _("Move"), wxPoint(1000,368),
wxSize(48,48), 0, wxDefaultValidator, _T("ID_BUTTON1"));
    StaticBitmap2 = new wxStaticBitmap(Panel1, ID_STATICBITMAP2,
wxBitmap(wxImage(_T("00.bmp")).Rescale(wxSize(774,776).GetWidth(),wxSize(774,776).Get
Height())), wxPoint(93,27), wxSize(774,776), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP2"));
    StaticBitmap1 = new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wxImage(_T("horse.bmp")).Rescale(wxSize(80,80).GetWidth(),wxSize(80,80).GetH
eight())), wxPoint(984,48), wxSize(80,80), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));
    Button3 = new wxButton(Panel1, ID_BUTTON3, _("Print"), wxPoint(16,40), wxSize(48,48),
0, wxDefaultValidator, _T("ID_BUTTON3"));
    StaticText1 = new wxStaticText(Panel1, ID_STATICTEXT1, _("***"), wxPoint(968,184),
wxDefaultSize, 0, _T("ID_STATICTEXT1"));
```

```
   TextCtrl1 = new wxTextCtrl(Panel1, ID_TEXTCTRL1, _("25"), wxPoint(992,288),
wxSize(48,21), 0, wxDefaultValidator, _T("ID_TEXTCTRL1"));
   TextCtrl2 = new wxTextCtrl(Panel1, ID_TEXTCTRL2, _("45"), wxPoint(992,328),
wxSize(48,21), 0, wxDefaultValidator, _T("ID_TEXTCTRL2"));
   StaticText2 = new wxStaticText(Panel1, ID_STATICTEXT2, _("***"), wxPoint(968,144),
wxDefaultSize, 0, _T("ID_STATICTEXT2"));
   Button5 = new wxButton(Panel1, ID_BUTTON5, _("Next"), wxPoint(16,120),
wxSize(48,48), 0, wxDefaultValidator, _T("ID_BUTTON5"));
   MenuBar1 = new wxMenuBar();
   Menu1 = new wxMenu();
   MenuItem1 = new wxMenuItem(Menu1, idMenuQuit, _("Quit\tAlt-F4"), _("Quit the
application"), wxITEM_NORMAL);
   Menu1->Append(MenuItem1);
   MenuBar1->Append(Menu1, _("&File"));
   SetMenuBar(MenuBar1);
   StatusBar1 = new wxStatusBar(this, ID_STATUSBAR1, 0, _T("ID_STATUSBAR1"));
   int __wxStatusBarWidths_1[1] = { -1 };
   int __wxStatusBarStyles_1[1] = { wxSB_NORMAL };
   StatusBar1->SetFieldsCount(1,__wxStatusBarWidths_1);
   StatusBar1->SetStatusStyles(1,__wxStatusBarStyles_1);
   SetStatusBar(StatusBar1);


Connect(ID_BUTTON1,wxEVT_COMMAND_BUTTON_CLICKED,(wxObjectEventFunction
)&_1grFrame::OnButton1Click2);

Connect(ID_BUTTON3,wxEVT_COMMAND_BUTTON_CLICKED,(wxObjectEventFunction
)&_1grFrame::OnButton3Click);

Connect(ID_BUTTON5,wxEVT_COMMAND_BUTTON_CLICKED,(wxObjectEventFunction
)&_1grFrame::OnButton5Click1);
   Panel1-
>Connect(wxEVT_PAINT,(wxObjectEventFunction)&_1grFrame::OnPanel1Paint,0,this);

Connect(idMenuQuit,wxEVT_COMMAND_MENU_SELECTED,(wxObjectEventFunction)&_
1grFrame::OnQuit);
   //*)
   this->SetTitle(wxT("Chess"));
}

_1grFrame::~_1grFrame()
{
   //(*Destroy(_1grFrame)
   //*)
}

        CChess qGame;
        CBoard mqGameBoard;

bool isblack (int j, int i){
   if      ((((j+1)%2==0)&&((i+1)%2==0))||(((j+1)%2==1)&&((i+1)%2==1))){return      true;}
else{return false;};
```

```
}
void _1grFrame::OnQuit(wxCommandEvent& event)
{
   Close();
}

void _1grFrame::OnPanel1Paint(wxPaintEvent& event)
{
}

void _1grFrame::OnButton1Click2(wxCommandEvent& event)
{
}

void _1grFrame::OnButton3Click(wxCommandEvent& event)
{
wxImage wgrcell(wxT("wcell.bmp"), wxBITMAP_TYPE_BMP);
wxImage bgrcell(wxT("bcell.bmp"), wxBITMAP_TYPE_BMP);

wxImage wwgrpawn(wxT("wwpawn.bmp"), wxBITMAP_TYPE_BMP);
wxImage bwgrpawn(wxT("bwpawn.bmp"), wxBITMAP_TYPE_BMP);
wxImage bbgrpawn(wxT("bbpawn.bmp"), wxBITMAP_TYPE_BMP);
wxImage wbgrpawn(wxT("wbpawn.bmp"), wxBITMAP_TYPE_BMP);

wxImage wwgrrook(wxT("wwrook.bmp"), wxBITMAP_TYPE_BMP);
wxImage bwgrrook(wxT("bwrook.bmp"), wxBITMAP_TYPE_BMP);
wxImage bbgrrook(wxT("bbrook.bmp"), wxBITMAP_TYPE_BMP);
wxImage wbgrrook(wxT("wbrook.bmp"), wxBITMAP_TYPE_BMP);

wxImage wwgrhorse(wxT("wwhorse.bmp"), wxBITMAP_TYPE_BMP);
wxImage bwgrhorse(wxT("bwhorse.bmp"), wxBITMAP_TYPE_BMP);
wxImage bbgrhorse(wxT("bbhorse.bmp"), wxBITMAP_TYPE_BMP);
wxImage wbgrhorse(wxT("wbhorse.bmp"), wxBITMAP_TYPE_BMP);

wxImage wwgrbishop(wxT("wwbishop.bmp"), wxBITMAP_TYPE_BMP);
wxImage bwgrbishop(wxT("bwbishop.bmp"), wxBITMAP_TYPE_BMP);
wxImage bbgrbishop(wxT("bbbishop.bmp"), wxBITMAP_TYPE_BMP);
wxImage wbgrbishop(wxT("wbbishop.bmp"), wxBITMAP_TYPE_BMP);

wxImage wwgrqueen(wxT("wwqueen.bmp"), wxBITMAP_TYPE_BMP);
wxImage bwgrqueen(wxT("bwqueen.bmp"), wxBITMAP_TYPE_BMP);
wxImage bbgrqueen(wxT("bbqueen.bmp"), wxBITMAP_TYPE_BMP);
wxImage wbgrqueen(wxT("wbqueen.bmp"), wxBITMAP_TYPE_BMP);

wxImage wwgrking(wxT("wwking.bmp"), wxBITMAP_TYPE_BMP);
wxImage bwgrking(wxT("bwking.bmp"), wxBITMAP_TYPE_BMP);
wxImage bbgrking(wxT("bbking.bmp"), wxBITMAP_TYPE_BMP);
wxImage wbgrking(wxT("wbking.bmp"), wxBITMAP_TYPE_BMP);

for ( j=0; j<8; j++){
for ( i=0; i<8; i++){
```

```
std::string z=mqGameBoard.obr(j,i,mqGameBoard.mqpaaBoard);

if (z=="0"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bgrcell), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="0"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wgrcell), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};

if (z=="WP"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wwgrpawn), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="BP"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bwgrpawn), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="WP"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wbgrpawn), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="BP"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bbgrpawn), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};

if (z=="WR"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wwgrrook), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="BR"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bwgrrook), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="WR"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wbgrrook), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="BR"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bbgrrook), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};

if (z=="WH"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wwgrhorse), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="BH"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bwgrhorse), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="WH"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wbgrhorse), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="BH"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bbgrhorse), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};

if (z=="WB"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wwgrbishop), wxPoint(smX+80*i,smY-j*80), wxSize(82,82),
wxSIMPLE_BORDER, _T("ID_STATICBITMAP1"));};
```

```
if (z=="BB"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bwgrbishop), wxPoint(smX+80*i,smY-j*80), wxSize(82,82),
wxSIMPLE_BORDER, _T("ID_STATICBITMAP1"));};
if (z=="WB"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wbgrbishop), wxPoint(smX+80*i,smY-j*80), wxSize(82,82),
wxSIMPLE_BORDER, _T("ID_STATICBITMAP1"));};
if (z=="BB"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bbgrbishop), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};

if (z=="WQ"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wwgrqueen), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="BQ"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bwgrqueen), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="WQ"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wbgrqueen), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="BQ"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bbgrqueen), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};

if (z=="WK"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wwgrking), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="BK"&&(!isblack (j,i))){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bwgrking), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="WK"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(wbgrking), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};
if (z=="BK"&&isblack (j,i)){cell[j][i]=new wxStaticBitmap(Panel1, ID_STATICBITMAP1,
wxBitmap(bbgrking), wxPoint(smX+80*i,smY-j*80), wxSize(82,82), wxSIMPLE_BORDER,
_T("ID_STATICBITMAP1"));};

            }
            }
}
void _1grFrame::OnButton5Click1(wxCommandEvent& event)
{
   qGame.GetNextMove(mqGameBoard.mqpaaBoard);
   qGame.AlternateTurn();
   if(qGame.IsGameOver(mqGameBoard.mqpaaBoard)){Close();};
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

ЛИСТИНГ ФАЙЛА "CAPiece.h"

```
#ifndef CAPIECE_H_INCLUDED
#define CAPIECE_H_INCLUDED
class CAPiece
{
public:
	CAPiece(char cColor) : mcColor(cColor) {}
	~CAPiece() {}
	virtual char GetPiece() = 0;
	char GetColor() {
		return mcColor;
	}
	bool IsLegalMove(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, CAPiece*
qpaaBoard[8][8]) {
		CAPiece* qpDest = qpaaBoard[iDestRow][iDestCol];
		if ((qpDest == 0) || (mcColor != qpDest->GetColor())) {
			return AreSquaresLegal(iSrcRow, iSrcCol, iDestRow, iDestCol,
qpaaBoard);
		}
		return false;
	}
private:
	virtual bool AreSquaresLegal(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol,
CAPiece* qpaaBoard[8][8]) = 0;
	char mcColor;
};
#endif
```

ПРИЛОЖЕНИЕ В

(обязательное)

ЛИСТИНГ ФАЙЛА "CFigure.h"

```
#ifndef CFIGURE_H_INCLUDED
#define CFIGURE_H_INCLUDED
#include "CAPiece.h"
class CPawn : public CAPiece
{
public:
	CPawn(char cColor) : CAPiece(cColor) {}
	~CPawn() {}
private:
	virtual char GetPiece() {
		return 'P';
	}
	bool AreSquaresLegal(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, CAPiece*
qpaaBoard[8][8]) {
		CAPiece* qpDest = qpaaBoard[iDestRow][iDestCol];

		if (qpDest == 0) {
			if (iSrcCol == iDestCol) {
				if (GetColor() == 'W') {
				   if(iSrcRow == 1){
		if (iDestRow == iSrcRow + 2) {
					return true;
		}
				   }
					if (iDestRow == iSrcRow + 1) {
						return true;
					}
				} else {
				   if(iSrcRow == 6){
		if (iDestRow == iSrcRow - 2) {
					return true;
		}
				   }
					if (iDestRow == iSrcRow - 1) {
						return true;
					}
				}
			}
		} else {
			if ((iSrcCol == iDestCol + 1) || (iSrcCol == iDestCol - 1)) {
				if (GetColor() == 'W') {
					if (iDestRow == iSrcRow + 1) {
						return true;
					}
				} else {
					if (iDestRow == iSrcRow - 1) {
```

```
                                        return true;
                                    }
                                }
                            }
                        }
                        return false;
                }
};

class CKnight : public CAPiece
{
public:
        CKnight(char cColor) : CAPiece(cColor) {}
        ~CKnight() {}
private:
        virtual char GetPiece() {
                return 'H';
        }
        bool AreSquaresLegal(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, CAPiece*
qpaaBoard[8][8]) {
                if ((iSrcCol == iDestCol + 1) || (iSrcCol == iDestCol - 1)) {
                        if ((iSrcRow == iDestRow + 2) || (iSrcRow == iDestRow - 2)) {
                                return true;
                        }
                }
                if ((iSrcCol == iDestCol + 2) || (iSrcCol == iDestCol - 2)) {
                        if ((iSrcRow == iDestRow + 1) || (iSrcRow == iDestRow - 1)) {
                                return true;
                        }
                }
                return false;
        }
};

class CBishop : public CAPiece
{
public:
        CBishop(char cColor) : CAPiece(cColor) {}
        ~CBishop() {}
private:
        virtual char GetPiece() {
                return 'B';
        }
        bool AreSquaresLegal(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, CAPiece*
qpaaBoard[8][8]) {
                if ((iDestCol - iSrcCol == iDestRow - iSrcRow) || (iDestCol - iSrcCol == iSrcRow
- iDestRow)) {
                        int iRowOffset = (iDestRow - iSrcRow > 0) ? 1 : -1;
                        int iColOffset = (iDestCol - iSrcCol > 0) ? 1 : -1;
                        int iCheckRow;
                        int iCheckCol;
```

```
                        for (iCheckRow = iSrcRow + iRowOffset, iCheckCol = iSrcCol +
iColOffset;
                                iCheckRow !=  iDestRow;
                                iCheckRow = iCheckRow + iRowOffset, iCheckCol = iCheckCol +
iColOffset)
                        {
                                if (qpaaBoard[iCheckRow][iCheckCol] != 0) {
                                        return false;
                                }
                        }
                        return true;
                }
                return false;
        }
};

class CRook : public CAPiece
{
public:
        CRook(char cColor) : CAPiece(cColor) {}
        ~CRook() {}
private:
        virtual char GetPiece() {
                return 'R';
        }
        bool AreSquaresLegal(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, CAPiece*
qpaaBoard[8][8]) {
                if (iSrcRow == iDestRow) {
                        int iColOffset = (iDestCol - iSrcCol > 0) ? 1 : -1;
                        for (int iCheckCol = iSrcCol + iColOffset; iCheckCol !=  iDestCol;
iCheckCol = iCheckCol + iColOffset) {
                                if (qpaaBoard[iSrcRow][iCheckCol] != 0) {
                                        return false;
                                }
                        }
                        return true;
                } else if (iDestCol == iSrcCol) {
                        int iRowOffset = (iDestRow - iSrcRow > 0) ? 1 : -1;
                        for (int iCheckRow = iSrcRow + iRowOffset; iCheckRow !=  iDestRow;
iCheckRow = iCheckRow + iRowOffset) {
                                if (qpaaBoard[iCheckRow][iSrcCol] != 0) {
                                        return false;
                                }
                        }
                        return true;
                }
                return false;
        }
};

class CQueen : public CAPiece
{
```

```
public:
	CQueen(char cColor) : CAPiece(cColor) {}
	~CQueen() {}
private:
	virtual char GetPiece() {
		return 'Q';
	}
	bool AreSquaresLegal(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, CAPiece*
qpaaBoard[8][8]) {
		if (iSrcRow == iDestRow) {
			int iColOffset = (iDestCol - iSrcCol > 0) ? 1 : -1;
			for (int iCheckCol = iSrcCol + iColOffset; iCheckCol !=  iDestCol;
iCheckCol = iCheckCol + iColOffset) {
				if (qpaaBoard[iSrcRow][iCheckCol] != 0) {
					return false;
				}
			}
			return true;
		} else if (iDestCol == iSrcCol) {
			int iRowOffset = (iDestRow - iSrcRow > 0) ? 1 : -1;
			for (int iCheckRow = iSrcRow + iRowOffset; iCheckRow !=  iDestRow;
iCheckRow = iCheckRow + iRowOffset) {
				if (qpaaBoard[iCheckRow][iSrcCol] != 0) {
					return false;
				}
			}
			return true;
		} else if ((iDestCol - iSrcCol == iDestRow - iSrcRow) || (iDestCol - iSrcCol ==
iSrcRow - iDestRow)) {
			int iRowOffset = (iDestRow - iSrcRow > 0) ? 1 : -1;
			int iColOffset = (iDestCol - iSrcCol > 0) ? 1 : -1;
			int iCheckRow;
			int iCheckCol;
			for (iCheckRow = iSrcRow + iRowOffset, iCheckCol = iSrcCol +
iColOffset;
				iCheckRow !=  iDestRow;
				iCheckRow = iCheckRow + iRowOffset, iCheckCol = iCheckCol +
iColOffset)
			{
				if (qpaaBoard[iCheckRow][iCheckCol] != 0) {
					return false;
				}
			}
			return true;
		}
		return false;
	}
};

class CKing : public CAPiece
{
public:
```

```
	CKing(char cColor) : CAPiece(cColor) {}
	~CKing() {}
private:
	virtual char GetPiece() {
		return 'K';
	}
	bool AreSquaresLegal(int iSrcRow, int iSrcCol, int iDestRow, int iDestCol, CAPiece*
qpaaBoard[8][8]) {
		int iRowDelta = iDestRow - iSrcRow;
		int iColDelta = iDestCol - iSrcCol;
		if (((iRowDelta >= -1) && (iRowDelta <= 1)) &&
			((iColDelta >= -1) && (iColDelta <= 1)))
		{
			return true;
		}
		return false;
	}
};
#endif
```

ПРИЛОЖЕНИЕ Г

(обязательное)

ЛИСТИНГ ФАЙЛА "CBoard.h"

```
#ifndef CBOARD_H_INCLUDED
#define CBOARD_H_INCLUDED
class CBoard
{
public:
	CBoard() {
		for (int iRow = 0; iRow < 8; ++iRow) {
			for (int iCol = 0; iCol < 8; ++iCol) {
				mqpaaBoard[iRow][iCol] = 0;
			}
		}
		for (int iCol = 0; iCol < 8; ++iCol) {
			mqpaaBoard[6][iCol] = new CPawn('B');
		}
		mqpaaBoard[7][0] = new CRook('B');
		mqpaaBoard[7][1] = new CKnight('B');
		mqpaaBoard[7][2] = new CBishop('B');
		mqpaaBoard[7][4] = new CKing('B');
		mqpaaBoard[7][3] = new CQueen('B');
		mqpaaBoard[7][5] = new CBishop('B');
		mqpaaBoard[7][6] = new CKnight('B');
		mqpaaBoard[7][7] = new CRook('B');
		for (int iCol = 0; iCol < 8; ++iCol) {
			mqpaaBoard[1][iCol] = new CPawn('W');
		}
		mqpaaBoard[0][0] = new CRook('W');
		mqpaaBoard[0][1] = new CKnight('W');
		mqpaaBoard[0][2] = new CBishop('W');
		mqpaaBoard[0][4] = new CKing('W');
		mqpaaBoard[0][3] = new CQueen('W');
		mqpaaBoard[0][5] = new CBishop('W');
		mqpaaBoard[0][6] = new CKnight('W');
		mqpaaBoard[0][7] = new CRook('W');
	}
	~CBoard() {
		for (int iRow = 0; iRow < 8; ++iRow) {
			for (int iCol = 0; iCol < 8; ++iCol) {
				delete mqpaaBoard[iRow][iCol];
				mqpaaBoard[iRow][iCol] = 0;
			}
		}
	}

std::string obr(int st,int str,CAPiece* qpaaBoard[8][8]){
		std::string z2="0";
		std::string res="";
```

```
        CAPiece* qpCurrPiece = qpaaBoard[st][str];
        if(qpCurrPiece != 0){
                char z1=qpCurrPiece->GetPiece();
                char z2=qpCurrPiece->GetColor();
                res+=z2;
                res+=z1;
                return res;
                }else{return z2;}
                }
          CAPiece* mqpaaBoard[8][8];
};
#endif
```

ПРИЛОЖЕНИЕ Д

(обязательное)

ЛИСТИНГ ФАЙЛА "CChess.h"

```
#ifndef CCHESS_H_INCLUDED
#define CCHESS_H_INCLUDED
class CChess
{
public:

std::string x;

        CChess() : mcPlayerTurn('W') {}
        ~CChess() {}
        void GetNextMove(/*int iStartMove, int iEndMove, */CAPiece* qpaaBoard[8][8]) {
                using namespace std;
                int temp;
                wxNumberEntryDialog dialog1(NULL,wxT("'s Move: "),mcPlayerTurn,
wxT("Numeric input"), 25, 11, 99);
                wxNumberEntryDialog dialog2(NULL,wxT("To: "),mcPlayerTurn,
wxT("Numeric input"), 45, 11, 99);

      wxMessageDialog dialogInvalidMove( NULL, wxT("Invalid
move!"),wxT(""),wxICON_INFORMATION);
      wxMessageDialog dialogAlarm( NULL,
wxT("Alarm!"),wxT(""),wxICON_INFORMATION);
      wxMessageDialog dialogColorRight( NULL, wxT("Color
right"),mcPlayerTurn,wxICON_INFORMATION);
      wxMessageDialog dialogMoveLegal( NULL, wxT("Move
legal"),wxT(""),wxICON_INFORMATION);
      wxMessageDialog dialogNoAlarm( NULL, wxT("No
alarm"),wxT(""),wxICON_INFORMATION);

      int iStartMove;
      int iEndMove;

                bool bValidMove               = false;
                bool z1;
                do {
        if (dialog1.ShowModal() == wxID_OK){ iStartMove = dialog1.GetValue();}else{goto
g1;};
                        int iStartRow = (iStartMove / 10) - 1;
                        int iStartCol = (iStartMove % 10) - 1;

                        if (dialog2.ShowModal() == wxID_OK){ iEndMove =
dialog2.GetValue();}else{goto g1;};
                        int iEndRow = (iEndMove / 10) - 1;
                        int iEndCol = (iEndMove % 10) - 1;
```

```
                    if ((iStartRow >= 0 && iStartRow <= 7) &&
                            (iStartCol >= 0 && iStartCol <= 7) &&
                            (iEndRow >= 0 && iEndRow <= 7) &&
                            (iEndCol >= 0 && iEndCol <= 7)) {
                            CAPiece* qpCurrPiece = qpaaBoard[iStartRow][iStartCol];
                            if ((qpCurrPiece != 0) && (qpCurrPiece->GetColor() ==
mcPlayerTurn)) {
                dialogColorRight.ShowModal();
                                    if (qpCurrPiece->IsLegalMove(iStartRow, iStartCol,
iEndRow, iEndCol, qpaaBoard)) {
                    dialogMoveLegal.ShowModal();
                                            CAPiece* qpTemp                                 =
qpaaBoard[iEndRow][iEndCol];
                                            qpaaBoard[iEndRow][iEndCol]             =
qpaaBoard[iStartRow][iStartCol];
                                            qpaaBoard[iStartRow][iStartCol]     = 0;

      int iKingRow;
                int iKingCol;
                for (int iRow = 0; iRow < 8; ++iRow) {
                        for (int iCol = 0; iCol < 8; ++iCol) {
                                if (qpaaBoard[iRow][iCol] != 0) {
                                        if (qpaaBoard[iRow][iCol]->GetColor() == mcPlayerTurn)
{
                                                if (qpaaBoard[iRow][iCol]->GetPiece() == 'K') {
                                                        iKingRow = iRow;
                                                        iKingCol = iCol;
                                                }
                                        }
                                }
                        }
                }
                for (int iRow = 0; iRow < 8; ++iRow) {
                        for (int iCol = 0; iCol < 8; ++iCol) {
                                if (qpaaBoard[iRow][iCol] != 0) {
                                        if (qpaaBoard[iRow][iCol]->GetColor() != mcPlayerTurn) {
                                                if (qpaaBoard[iRow][iCol]->IsLegalMove(iRow,
iCol, iKingRow, iKingCol, qpaaBoard)) {
                                                        z1=true;
                                                        goto m111;
                                                }
                                        }
                                }
                        }
                }
      z1=false;
                                            m111:if (!z1) {
                                               dialogNoAlarm.ShowModal();
                                                    delete qpTemp;
                                                    bValidMove = true;
                                            } else {
```

```
                                        qpaaBoard[iStartRow][iStartCol] =
qpaaBoard[iEndRow][iEndCol];
                                        qpaaBoard[iEndRow][iEndCol]          =
qpTemp;
                                        dialogAlarm.ShowModal();
                                    }
                                }
                            }
                        }
                        if (!bValidMove) {
                dialogInvalidMove.ShowModal();
                        }
                    } while (!bValidMove);
                    goto g2;
                    g1:AlternateTurn();
                    g2:temp=0;
            }

            void AlternateTurn() {
                    mcPlayerTurn = (mcPlayerTurn == 'W') ? 'B' : 'W';
            }

               bool IsGameOver(CAPiece* qpaaBoard[8][8]) {
               using namespace std;
               bool t1,t2;
                    bool bCanMove=false;
                    // просмотреть все поле
                    for (int iRow = 0; iRow < 8; ++iRow) {
                            for (int iCol = 0; iCol < 8; ++iCol) {
                                    if (qpaaBoard[iRow][iCol] != 0) {
                                            if (qpaaBoard[iRow][iCol]->GetColor() == mcPlayerTurn)
{
                                                    for (int iMoveRow = 0; iMoveRow < 8;
++iMoveRow) {
                                                            for (int iMoveCol = 0; iMoveCol < 8;
++iMoveCol) {
                                                                    if (qpaaBoard[iRow][iCol]-
>IsLegalMove(iRow, iCol, iMoveRow, iMoveCol, qpaaBoard)) {
CAPiece* qpTemp= qpaaBoard[iMoveRow][iMoveCol];
qpaaBoard[iMoveRow][iMoveCol]  = qpaaBoard[iRow][iCol];
 [iRow][iCol] = 0;
                    int iKingRow;
                    int iKingCol;
                    for (int iRow = 0; iRow < 8; ++iRow) {
                            for (int iCol = 0; iCol < 8; ++iCol) {
                                    if (qpaaBoard[iRow][iCol] != 0) {
                                            if (qpaaBoard[iRow][iCol]->GetColor() == mcPlayerTurn)
{
                                                    if (qpaaBoard[iRow][iCol]->GetPiece() == 'K') {
                                                            iKingRow = iRow;
                                                            iKingCol = iCol;
                                                    }
```

```
                                        }
                                    }
                                }
                            }
                            for (int iRow = 0; iRow < 8; ++iRow) {
                                for (int iCol = 0; iCol < 8; ++iCol) {
                                    if (qpaaBoard[iRow][iCol] != 0) {
                                        if (qpaaBoard[iRow][iCol]->GetColor() != mcPlayerTurn) {
                                            if (qpaaBoard[iRow][iCol]->IsLegalMove(iRow,
iCol, iKingRow, iKingCol, qpaaBoard)) {
                    bCanMove= true;
                    goto m3;
                                            }
                                        }
                                    }
                                }
                            }
        bCanMove=false;
m3:qpaaBoard[iRow][iCol] = qpaaBoard[iMoveRow][iMoveCol];

            qpaaBoard[iMoveRow][iMoveCol]  = qpTemp;
                                                                    if (!bCanMove) {
                                                                        t2=true;
                                                                        goto m4;
                                                                    }

                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
            t2=false;
                            m4:if (!t2) {
        int iKingRow;
                            int iKingCol;
                            for (int iRow = 0; iRow < 8; ++iRow) {
                                for (int iCol = 0; iCol < 8; ++iCol) {
                                    if (qpaaBoard[iRow][iCol] != 0) {
                                        if (qpaaBoard[iRow][iCol]->GetColor() == mcPlayerTurn)
{
                                            if (qpaaBoard[iRow][iCol]->GetPiece() == 'K') {
                                                iKingRow = iRow;
                                                iKingCol = iCol;
                                            }
                                        }
                                    }
                                }
                            }
                            for (int iRow = 0; iRow < 8; ++iRow) {
                                for (int iCol = 0; iCol < 8; ++iCol) {
```

```
                                if (qpaaBoard[iRow][iCol] != 0) {
                                        if (qpaaBoard[iRow][iCol]->GetColor() != mcPlayerTurn) {
                                                if (qpaaBoard[iRow][iCol]->IsLegalMove(iRow,
iCol, iKingRow, iKingCol, qpaaBoard)) {
                        t1= true;
                        goto m2;
                                                }
                                        }
                                }
                        }
                }
        t1=false;
                        m2:if (t1) {
                                AlternateTurn();
                                if (mcPlayerTurn=='W'){ wxMessageDialog
dialog(NULL,wxT("Checkmate! W Wins"),wxT(""),wxICON_INFORMATION);
                switch(dialog.ShowModal()){
                case wxID_YES:
                break;
                }
                                }if (mcPlayerTurn=='B') {wxMessageDialog
dialog(NULL,wxT("Checkmate! B Wins"),wxT(""),wxICON_INFORMATION);
                switch(dialog.ShowModal()){
                case wxID_YES:
                break;
                }
              }
                        } else {
                           wxMessageDialog dialog( NULL,
wxT("Stalemate"),wxT(""),wxICON_INFORMATION);
                switch(dialog.ShowModal()){
                case wxID_YES:
                break;
                }
                        }
                   }
                   return !t2;
          }
private:
          //CBoard mqGameBoard;
          char mcPlayerTurn;
};
#endif
```