Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ИНДИВИДУАЛЬНОЕ ДОМАШНЕЕ ЗАДАНИЕ

По объектно-ориентированному программированию

«Разработка графического редактора на языке C++ с применением механизмов ООП»

Студент \_\_\_\_\_\_\_\_\_ Лучинкин К.О.

Группа ПМ-21-1

Руководитель

к.т.н., доцент каф. АСУ \_\_\_\_\_\_\_\_\_ Кургасов В. В.

Липецк 2022 г

**Цель работы:** закрепить навыки использования механизмов ООП на примере реализации графического редактора.

**Задание кафедры:** реализовать на языке C++ редактор графической схемы. В ходе выполненной работы обязательно применение объектно-ориентированных возможностей языка C++: наследования и динамического полиморфизма. Каждый тип элемента схемы должен быть представлен в программе в виде отдельного класса, который наследован от базового класса «графический элемент» (имеющего чисто виртуальную функцию прорисовки). Также необходим один класс «поле рисования», который содержит все графические элементы и отвечает за вызов функций прорисовки. Хранение графических элементов осуществляется с использованием контейнеров стандартной библиотеки C++.

**Вариант 11:** Редактор позиций на шахматной доске.

**Исходный код**

Chess\_board.cpp

#include "Chess\_board.h"

using namespace System;

using namespace System::Windows::Forms;

[STAThreadAttribute]

void main(array<String^>^ args) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Chess::Chess\_board form;

Application::Run(% form);

}

Element.h

#pragma once

using namespace System::Drawing;

ref class Element abstract

{

public:

int type;

virtual void draw(Graphics^ g) = 0;

virtual Element^ find(int x, int y) = 0;

};

DrawingField.h

#pragma once

#include <cliext/list>

#include "Element.h"

ref class DrawingField

{

private:

cliext::list<Element^> forms;

public:

void addElement(Element^ form);

void drawElement(Graphics^ g);

Element^ findElement(int x, int y);

void deleteElement(Element^ form);

void clear();

};

DrawingField.cpp

#include "DrawingField.h"

void DrawingField::addElement(Element^ form)

{

forms.push\_front(form);

}

void DrawingField::drawElement(Graphics^ g)

{

if (!forms.empty()) {

for each (Element ^ form in forms) {

form->draw(g);

}

}

}

Element^ DrawingField::findElement(int x, int y)

{

Element^ result = nullptr;

for each (Element ^ form in forms)

if (form->find(x, y) != nullptr)

{

result = form;

forms.remove(form);

break;

}

return result;

}

void DrawingField::clear() {

forms.clear();

}

void DrawingField::deleteElement(Element^ form)

{

forms.remove(form);

}

Black\_Kon.cpp

#include "Black\_Kon.h"

using namespace System::Drawing;

Black\_Kon::Black\_Kon(Point point) {

this->p = point;

type = 9;

}

Point Black\_Kon::getBlack\_Kon() {

return p;

}

void Black\_Kon::setBlack\_Kon(Point point) {

this->p = point;

}

void Black\_Kon::draw(Graphics^ g) {

Image^ img = Image::FromFile("Black\_Kon.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Black\_Kon::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Black\_Kon.h

#pragma once

#include "Element.h"

ref class Black\_Kon :

public Element

{

private:

Point p;

int width, height;

public:

Black\_Kon(Point point);

Point getBlack\_Kon();

void setBlack\_Kon(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Black\_Korol.cpp

#include "Black\_Korol.h"

using namespace System::Drawing;

Black\_Korol::Black\_Korol(Point point) {

this->p = point;

type = 10;

}

Point Black\_Korol::getBlack\_Korol() {

return p;

}

void Black\_Korol::setBlack\_Korol(Point point) {

this->p = point;

}

void Black\_Korol::draw(Graphics^ g) {

Image^ img = Image::FromFile("Black\_Korol.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Black\_Korol::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Black\_Korol.h

#pragma once

#include "Element.h"

ref class Black\_Korol :

public Element

{

private:

Point p;

int width, height;

public:

Black\_Korol(Point point);

Point getBlack\_Korol();

void setBlack\_Korol(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Black\_Koroleva.cpp

#include "Black\_Koroleva.h"

using namespace System::Drawing;

Black\_Koroleva::Black\_Koroleva(Point point) {

this->p = point;

type = 11;

}

Point Black\_Koroleva::getBlack\_Koroleva() {

return p;

}

void Black\_Koroleva::setBlack\_Koroleva(Point point) {

this->p = point;

}

void Black\_Koroleva::draw(Graphics^ g) {

Image^ img = Image::FromFile("Black\_Koroleva.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Black\_Koroleva::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Black\_Koroleva.h

#pragma once

#include "Element.h"

ref class Black\_Koroleva :

public Element

{

private:

Point p;

int width, height;

public:

Black\_Koroleva(Point point);

Point getBlack\_Koroleva();

void setBlack\_Koroleva(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Black\_Ladya.cpp

#include "Black\_Ladya.h"

using namespace System::Drawing;

Black\_Ladya::Black\_Ladya(Point point) {

this->p = point;

type = 12;

}

Point Black\_Ladya::getBlack\_Ladya() {

return p;

}

void Black\_Ladya::setBlack\_Ladya(Point point) {

this->p = point;

}

void Black\_Ladya::draw(Graphics^ g) {

Image^ img = Image::FromFile("Black\_Ladya.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Black\_Ladya::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Black\_Ladya.h

#pragma once

#include "Element.h"

ref class Black\_Ladya :

public Element

{

private:

Point p;

int width, height;

public:

Black\_Ladya(Point point);

Point getBlack\_Ladya();

void setBlack\_Ladya(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Black\_Peshka.cpp

#include "Black\_Peshka.h"

using namespace System::Drawing;

Black\_Peshka::Black\_Peshka(Point point) {

this->p = point;

type = 13;

}

Point Black\_Peshka::getBlack\_Peshka() {

return p;

}

void Black\_Peshka::setBlack\_Peshka(Point point) {

this->p = point;

}

void Black\_Peshka::draw(Graphics^ g) {

Image^ img = Image::FromFile("Black\_Peshka.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Black\_Peshka::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Black\_Peshka.h

#pragma once

#include "Element.h"

ref class Black\_Peshka :

public Element

{

private:

Point p;

int width, height;

public:

Black\_Peshka(Point point);

Point getBlack\_Peshka();

void setBlack\_Peshka(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Black\_Slon.cpp

#include "Black\_Slon.h"

using namespace System::Drawing;

Black\_Slon::Black\_Slon(Point point) {

this->p = point;

type = 14;

}

Point Black\_Slon::getBlack\_Slon() {

return p;

}

void Black\_Slon::setBlack\_Slon(Point point) {

this->p = point;

}

void Black\_Slon::draw(Graphics^ g) {

Image^ img = Image::FromFile("Black\_Slon.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Black\_Slon::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Black\_Slon.h

#pragma once

#include "Element.h"

ref class Black\_Slon :

public Element

{

private:

Point p;

int width, height;

public:

Black\_Slon(Point point);

Point getBlack\_Slon();

void setBlack\_Slon(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Kon.cpp

#include "Kon.h"

using namespace System::Drawing;

Kon::Kon(Point point) {

this->p = point;

type = 2;

}

Point Kon::getKon() {

return p;

}

void Kon::setKon(Point point) {

this->p = point;

}

void Kon::draw(Graphics^ g) {

Image^ img = Image::FromFile("Kon.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Kon::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Kon.h

#pragma once

#include "Element.h"

ref class Kon :

public Element

{

private:

Point p;

int width, height;

public:

Kon(Point point);

Point getKon();

void setKon(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Korol.cpp

#include "Korol.h"

using namespace System::Drawing;

Korol::Korol(Point point) {

this->p = point;

type = 3;

}

Point Korol::getKorol() {

return p;

}

void Korol::setKorol(Point point) {

this->p = point;

}

void Korol::draw(Graphics^ g) {

Image^ img = Image::FromFile("Korol.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Korol::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Korol.h

#pragma once

#include "Element.h"

ref class Korol :

public Element

{

private:

Point p;

int width, height;

public:

Korol(Point point);

Point getKorol();

void setKorol(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Koroleva.cpp

#include "Koroleva.h"

using namespace System::Drawing;

Koroleva::Koroleva(Point point) {

this->p = point;

type = 4;

}

Point Koroleva::getKoroleva() {

return p;

}

void Koroleva::setKoroleva(Point point) {

this->p = point;

}

void Koroleva::draw(Graphics^ g) {

Image^ img = Image::FromFile("Koroleva.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Koroleva::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Koroleva.h

#pragma once

#include "Element.h"

ref class Koroleva :

public Element

{

private:

Point p;

int width, height;

public:

Koroleva(Point point);

Point getKoroleva();

void setKoroleva(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Ladya.cpp

#include "Ladya.h"

using namespace System::Drawing;

Ladya::Ladya(Point point) {

this->p = point;

type = 5;

}

Point Ladya::getLadya() {

return p;

}

void Ladya::setLadya(Point point) {

this->p = point;

}

void Ladya::draw(Graphics^ g) {

Image^ img = Image::FromFile("Ladya.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Ladya::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Ladya.h

#pragma once

#include "Element.h"

ref class Ladya :

public Element

{

private:

Point p;

int width, height;

public:

Ladya(Point point);

Point getLadya();

void setLadya(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Peshka.cpp

#include "Peshka.h"

using namespace System::Drawing;

Peshka::Peshka(Point point) {

this->p = point;

type = 6;

}

Point Peshka::getPeshka() {

return p;

}

void Peshka::setPeshka(Point point) {

this->p = point;

}

void Peshka::draw(Graphics^ g) {

Image^ img = Image::FromFile("Peshka.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Peshka::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Peshka.h

#pragma once

#include "Element.h"

ref class Peshka :

public Element

{

private:

Point p;

int width, height;

public:

Peshka(Point point);

Point getPeshka();

void setPeshka(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Slon.cpp

#include "Slon.h"

using namespace System::Drawing;

Slon::Slon(Point point) {

this->p = point;

type = 8;

}

Point Slon::getSlon() {

return p;

}

void Slon::setSlon(Point point) {

this->p = point;

}

void Slon::draw(Graphics^ g) {

Image^ img = Image::FromFile("Slon.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Slon::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Slon.h

#pragma once

#include "Element.h"

ref class Slon :

public Element

{

private:

Point p;

int width, height;

public:

Slon(Point point);

Point getSlon();

void setSlon(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

Pole.cpp

#include "Pole.h"

using namespace System::Drawing;

Pole::Pole(Point point) {

this->p = point;

type = 7;

}

Point Pole::getPole() {

return p;

}

void Pole::setPole(Point point) {

this->p = point;

}

void Pole::draw(Graphics^ g) {

Image^ img = Image::FromFile("Pole.png");

width = img->Width;

height = img->Height;

g->DrawImage(img, p);

}

Element^ Pole::find(int x, int y)

{

if (x >= p.X && x <= p.X + width) {

if (y >= p.Y && y <= p.Y + height)

return this;

}

return nullptr;

}

Pole.h

#pragma once

#include "Element.h"

ref class Pole :

public Element

{

private:

Point p;

int width, height;

public:

Pole(Point point);

Point getPole();

void setPole(Point point);

virtual void draw(Graphics^ g) override;

virtual Element^ find(int x, int y) override;

};

**Результаты работы алгоритма**

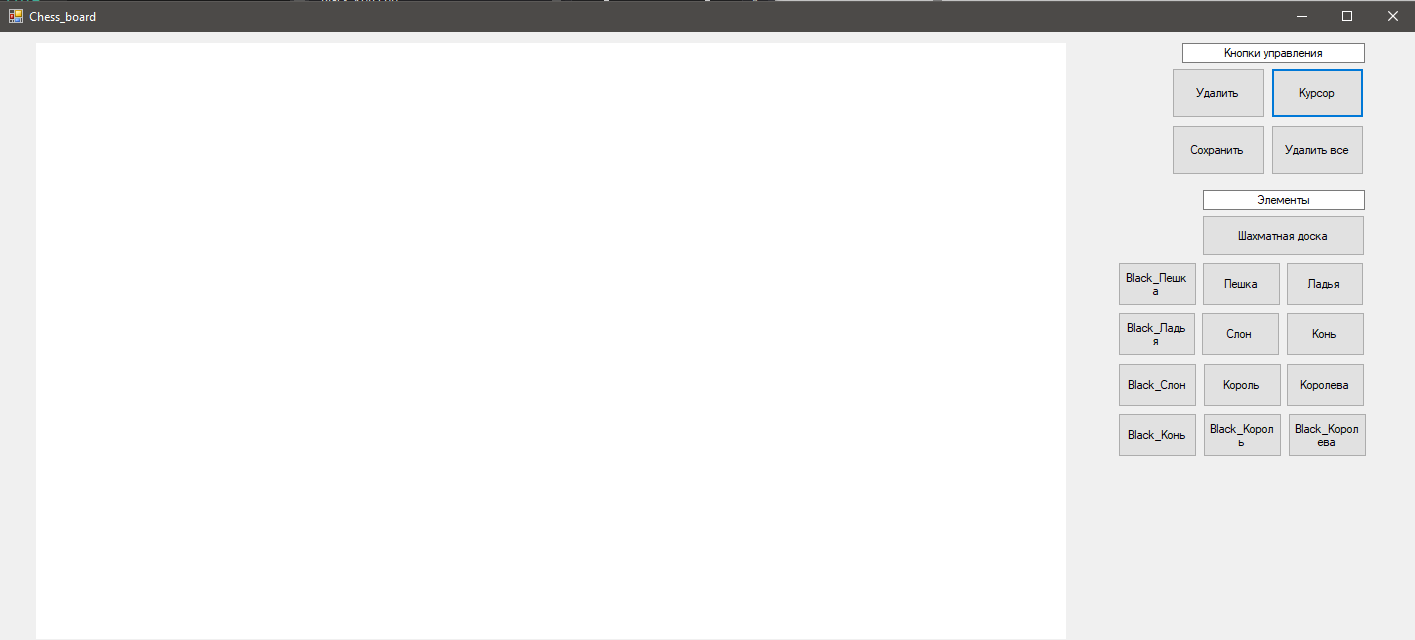


Рисунок 1. Начальное окно программы.

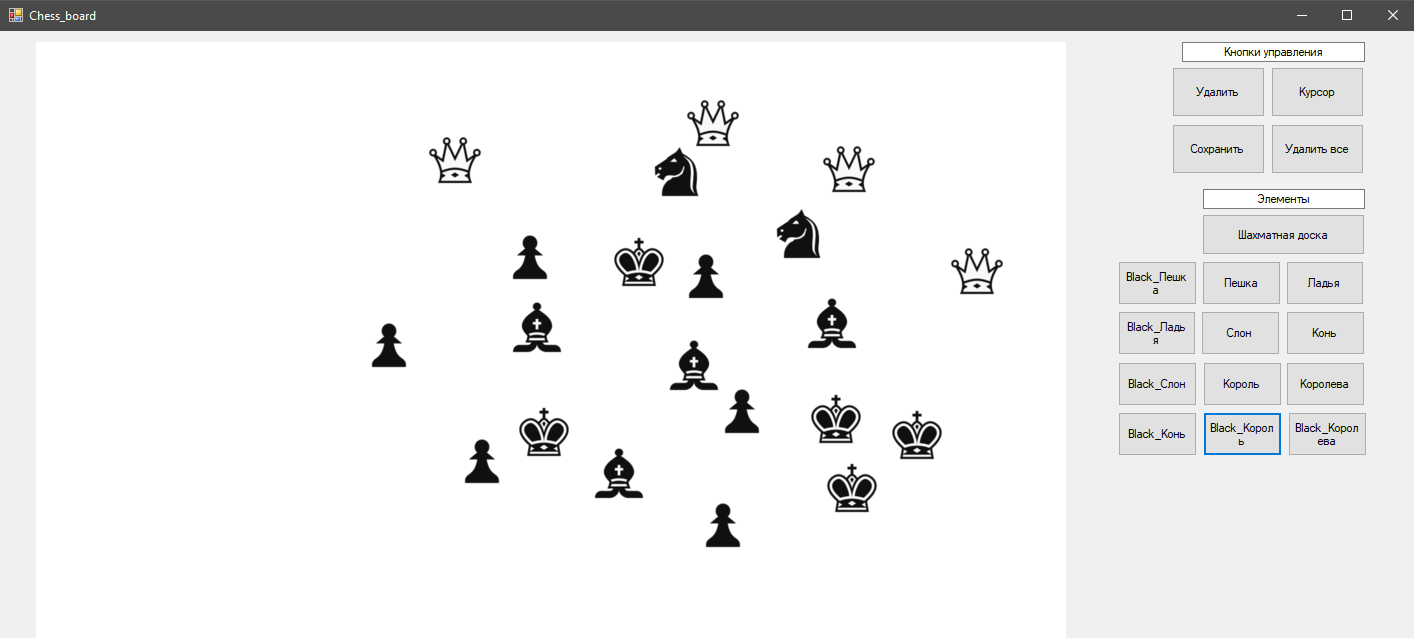


Рисунок 2. Пример расстановки фигур.

Для того чтобы поставить фишку на поле нужно выбрать её и кликнуть ЛКМ на поле

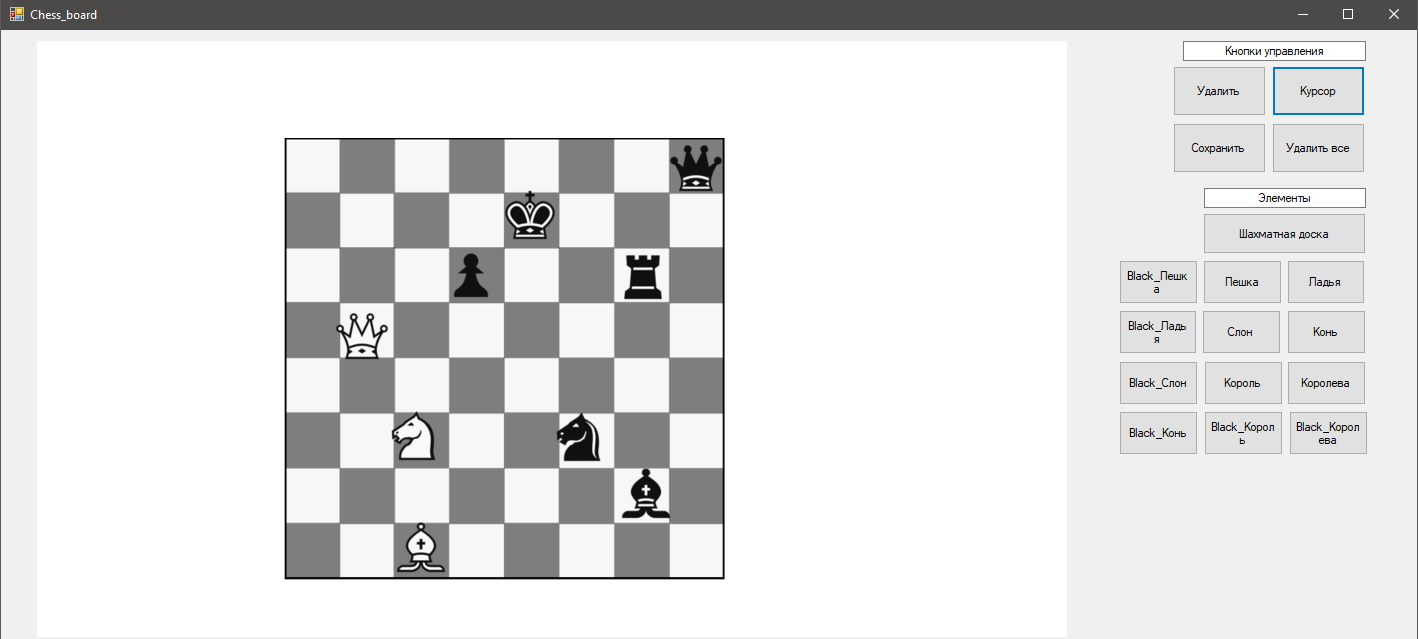


Рисунок 3. Пример расстановки с шахматной доской.

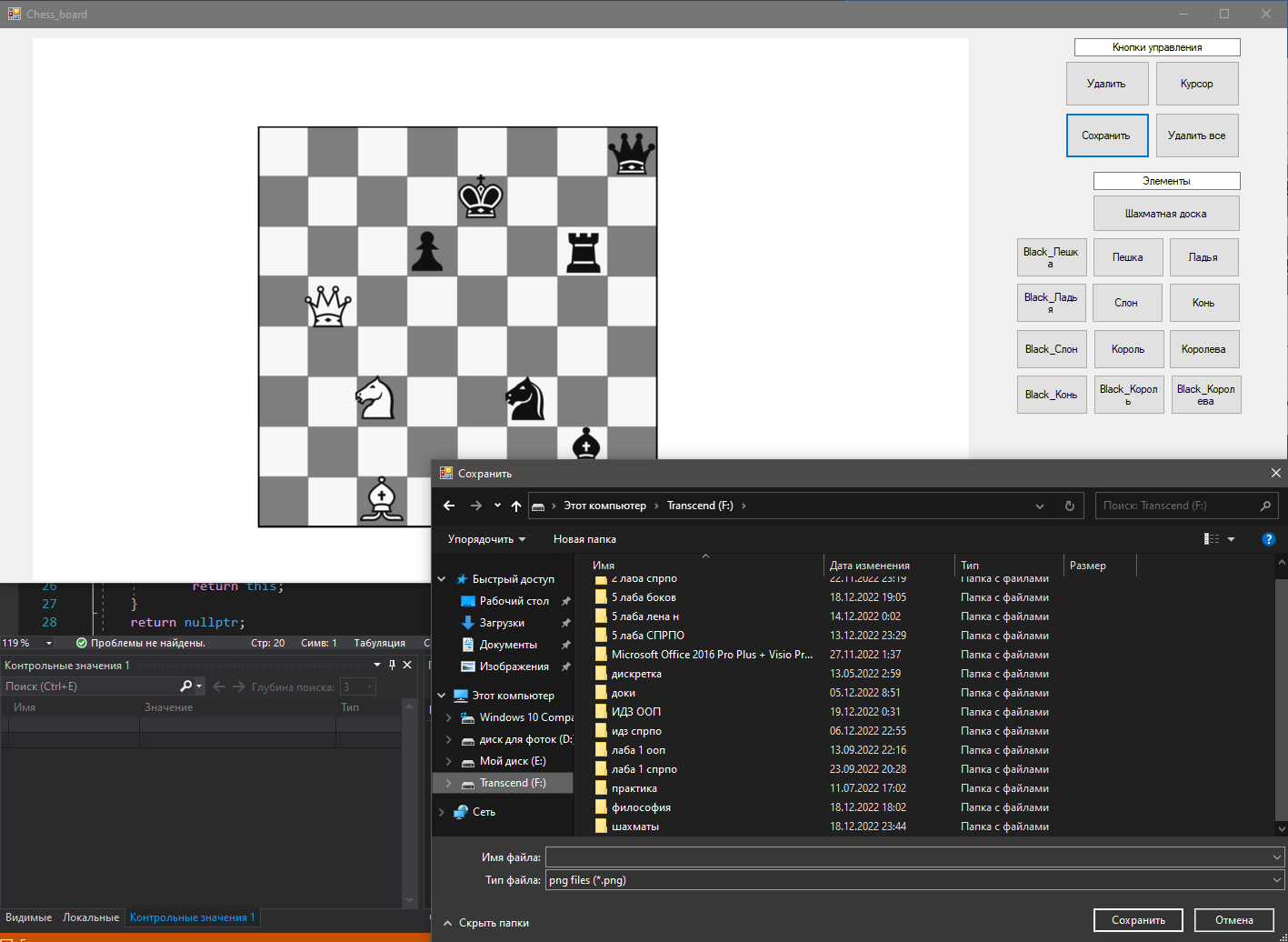


Рисунок 4. Пример сохранения изображения.

После того, как пользователь расставит фигуры, он может сохранить изображение в удобную для него папку

**Ответы на контрольные вопросы**

**1. Зачем нужен перегруженный оператор присваивания?**

Перегруженный оператор присваивания используется для присваивания объектов одного конкретного класса друг другу, если объекты класса не являются обычными типами данных.

**2. Зачем нужен механизм наследования?**

Наследование – это механизм создания нового класса на основе уже существующего. Основное назначение механизма наследования — повторное использование кодов, так как большинство используемых типов данных являются вариантами друг друга, и писать для каждого свой класс нецелесообразно.

Объекты разных классов и сами классы могут находиться в отношении наследования, при котором формируется иерархия объектов, соответствующая заранее предусмотренной иерархии классов.

**3. Зачем используются модификаторы при наследовании классов? Какие это модификаторы?**

Существуют модификаторы доступа и наследования - public, private, protected.

В модификаторах доступа:

Public – доступ открыт всем, кто видит определение данного класса.

Protected – доступ открыт классам, производным от данного. То есть, производные классы получают свободный доступ к таким свойствам или метода. Все другие классы такого доступа не имеют.

Private – доступ открыт самому классу (т.е. функциям-членам данного класса) и друзьям (friend) данного класса – как функциям, так и классам. Однако производные классы не получают доступа к этим данным совсем. И все другие классы такого доступа не имеют.

В C++ существует public-наследование, private-наследование и protected-наследование. В зависимости от того, какой тип используется, изменяется доступ к членам базового класса для клиентов производного.

**4. Зачем нужен механизм полиморфизма?**

Полиморфизм – это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных. Тип данных, который используется при вызове функции, определяет, какая конкретная версия функции действительно выполняется. В С++ можно использовать одно имя функции для множества различных действий. Это называется перегрузкой функций. В более общем смысле, концепцией полиморфизма является идея «один интерфейс, множество методов».

**5. Что понимается под динамическим полиморфизме?**

Динамический полиморфизм предстает перед нами в форме классов с виртуальными функциями и объектов, работа с которыми осуществляется косвенно – через указатели или ссылки, в то время как статический полиморфизм включает шаблоны классов и функций.

**6. Что такое интерфейс класса?**

Интерфейс – это класс, который не имеет переменных-членов, и все методы которого являются чистыми виртуальными функциями.

**7. Зачем нужен чисто виртуальный метод?**

Чисто виртуальный метод является методом, который объявляется в базовом классе, но не имеет в нем определения. Поскольку она не имеет определения, то есть тела в этом базовом классе, то всякий производный класс обязан иметь свою собственную версию определения данного метода. При введении чисто виртуальной функции в производном классе обязательно необходимо определить свою собственную реализацию этой функции. Если класс не будет содержать определения этой функции, то компилятор выдаст ошибку.

**8. Какой класс называется абстрактным?**

Если класс имеет хотя бы одну чисто виртуальную функцию, то такой класс называется абстрактным. Важной особенностью абстрактных классов является то, что он не имеет объектов. Вместо этого абстрактный класс служит в качестве базового для других производных классов. Причина, по которой абстрактный класс не может быть использован для объявления объекта, заключается в том, что одна или несколько его функций-членов не имеют определения. Тем не менее, даже если базовый класс является абстрактным, все равно можно объявлять указатели или ссылки на него, с помощью которых затем поддерживается полиморфизм времени исполнения.

**Вывод:** в данном индивидуальном задании я закрепил навыки ООП и создал графический редактор позиций фигур на шахматной доске.