

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное  
учреждение высшего образования  
«Южно-Уральский государственный университет  
(национальный исследовательский университет)»  
Институт естественных и точных наук  
Факультет математики, механики и компьютерных технологий  
Кафедра прикладной математики и программирования

Разработка программы «Шашки»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ  
по дисциплине «Компьютерная графика»  
ЮУрГУ–01.03.02.2021.152.ПЗ КП

Автор работы,  
студент группы ЕТ-412  
\_\_\_\_\_ / Д. П. Чумаков  
« \_\_\_\_ » \_\_\_\_\_ 2021 г.

Руководитель работы,  
доцент кафедры ПМиП  
\_\_\_\_\_ / Е.Ю. Алексеева  
« \_\_\_\_ » \_\_\_\_\_ 2021 г.

Работа защищена с оценкой  
\_\_\_\_\_  
« \_\_\_\_ » \_\_\_\_\_ 2021 г.

Челябинск 2021

## АННОТАЦИЯ

Чумаков Д.П. Разработка программы «Шашки». – Челябинск: ЮУрГУ, ЕТ-412, 2020. – 56 с., 28 ил., библиогр. список – 5 наим., 3 прил.

Целью работы является разработка игры «Шашки». В разделе 1 приведены требования к интерфейсу и функционалу программы, а также схемы программного меню, игрового поля и вспомогательных окон. В разделе 2 описана математическая модель игры и структуры данных, которые используются в программной реализации. В разделе 3 представлены схемы алгоритмов, в том числе алгоритм работы меню, содержательный алгоритм игры, алгоритм рисования элементов, алгоритм поиска возможных ходов. В разделе 4 приведено описание программных модулей, функций, структур данных и глобальных переменных. Текст программы, руководство пользователя и результаты выполнения программы приведены в приложениях.

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	4
1 ПОСТАНОВКА ЗАДАЧИ.....	5
2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ.....	8
3 ОПИСАНИЕ АЛГОРИТМА .....	10
4 ОПИСАНИЕ ПРОГРАММЫ .....	13
ЗАКЛЮЧЕНИЕ .....	14
БИБЛИОГРАФИЧЕСКИЙ СПИСОК .....	15
ПРИЛОЖЕНИЕ 1 Текст программы.....	16
ПРИЛОЖЕНИЕ 2 Руководство пользователя.....	47
ПРИЛОЖЕНИЕ 3 Результат выполнения программы.....	53

## ВВЕДЕНИЕ

Цель работы: разработать приложение игры «Шашки» в среде визуального программирования Microsoft Visual Studio 2019 с использованием графической библиотеки OpenTK.

Задачи работы:

- привести постановку задачи;
- провести анализ предметной области;
- описать параметры внешней среды, используемые пользовательские классы;
- привести схему алгоритма решения;
- определить способы визуализации элементов игры;
- привести инструкцию по применению данной программы.

# 1 ПОСТАНОВКА ЗАДАЧИ

Необходимо разработать компьютерную программу, реализующую игру «Шашки», на языке программирования C#. Среда разработки – Microsoft Visual Studio 2019.

Главное окно содержит поле для рисования 3d и главное меню.

Управление шашками производится с помощью мыши, для приближения или отдаления от доски используется колесо мыши. Для поворота вокруг оси  $Ox$ ,  $Oy$ ,  $Oz$  используются кнопки стрелок вверх и вниз, влево и вправо, латинские буквы «A» и «D» соответственно. Ось  $Oz$  направлена от экрана к пользователю.

После запуска приложения на экране появляется главное окно (рисунок 1.1). В правой части главного окна находится текущий счет игроков. Оси, изображенные на рисунке, в программе не рисуются и приведены здесь для наглядности.

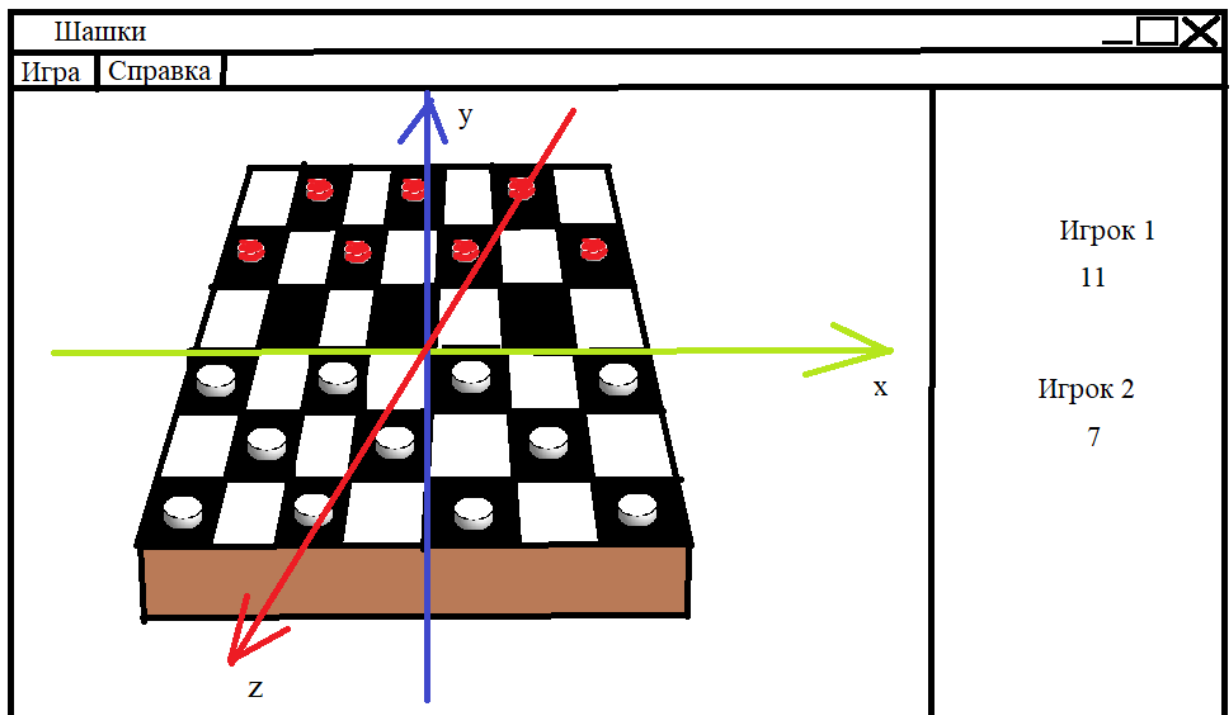


Рисунок 1.1 – Главное окно

При нажатии на кнопку «Новая игра», которая находится в меню «Игра» появится модальное окно для ввода имен игроков. (Рисунок 1.2).

При нажатии кнопки «Справка» выводится окно с руководством пользования (рисунок 1.3).

При нажатии кнопки «О программе» выводится окно с информацией о программе и ее авторе (рисунок 1.4).

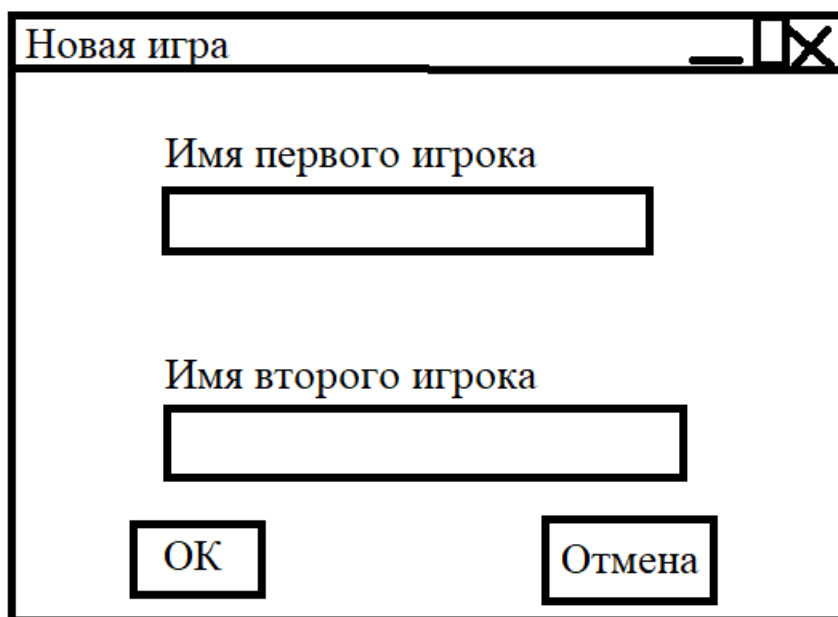


Рисунок 1.2 – Новая игра

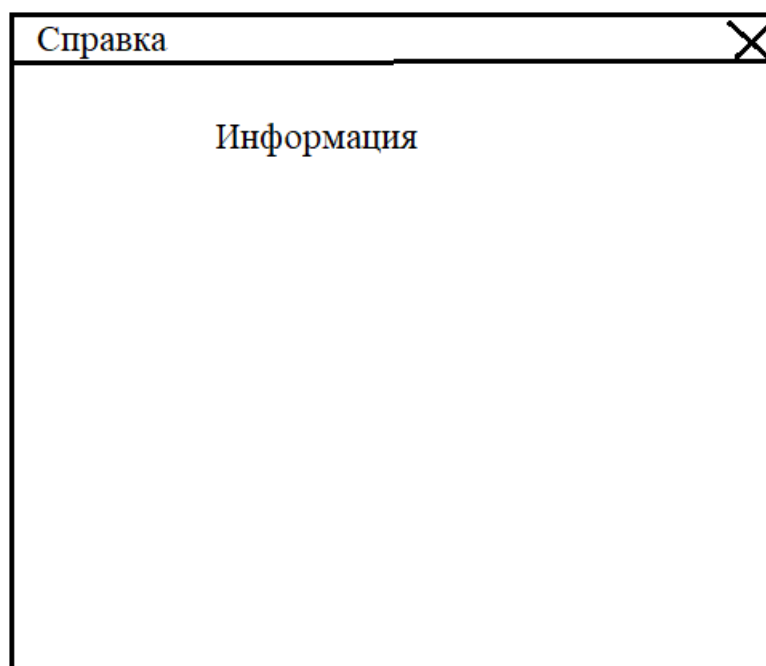


Рисунок 1.3 – Справка

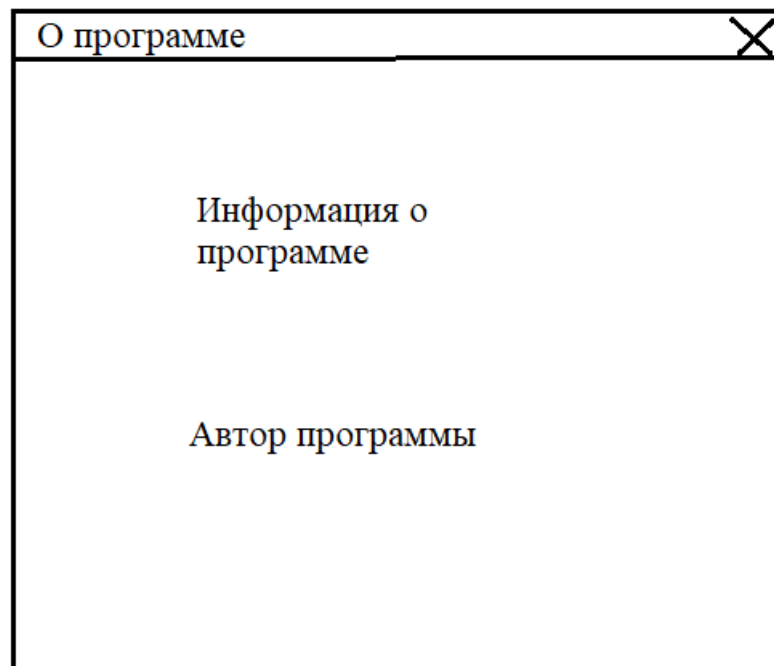


Рисунок 1.4 – О программе

## 2 МАТЕМАТИЧЕСКАЯ МОДЕЛЬ

На главном окне отображаются только шахматная доска и шашки. С математической точки зрения расположение шашек на доске можно представить в виде матрицы  $F$  размера  $8 \times 8$ , каждый элемент которой может принимать только одно из пяти возможных значений: 0 – клетка свободна, 1 – на клетке белая пешка, 2 – на клетке черная пешка, 3 – на клетке белая дамка, 4 – на клетке черная дамка. На рисунке 2.1 показан пример расположения шашек, а в формуле (1) представлена соответствующая матрица  $F$ .



Рисунок 2.1 – Пример расположения шашек

$$F = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 3 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad (1)$$

С графической точки зрения, доска представляет собой прямоугольный параллелепипед, а фишки – цилиндры. На рисунках 2.2 и 2.3 изображена доска размером  $3 \times 3$ , и приведены следующие обозначения:  $a$  – длина и ширина доски,  $c$  – толщина доски,  $r$  – радиус шашки.



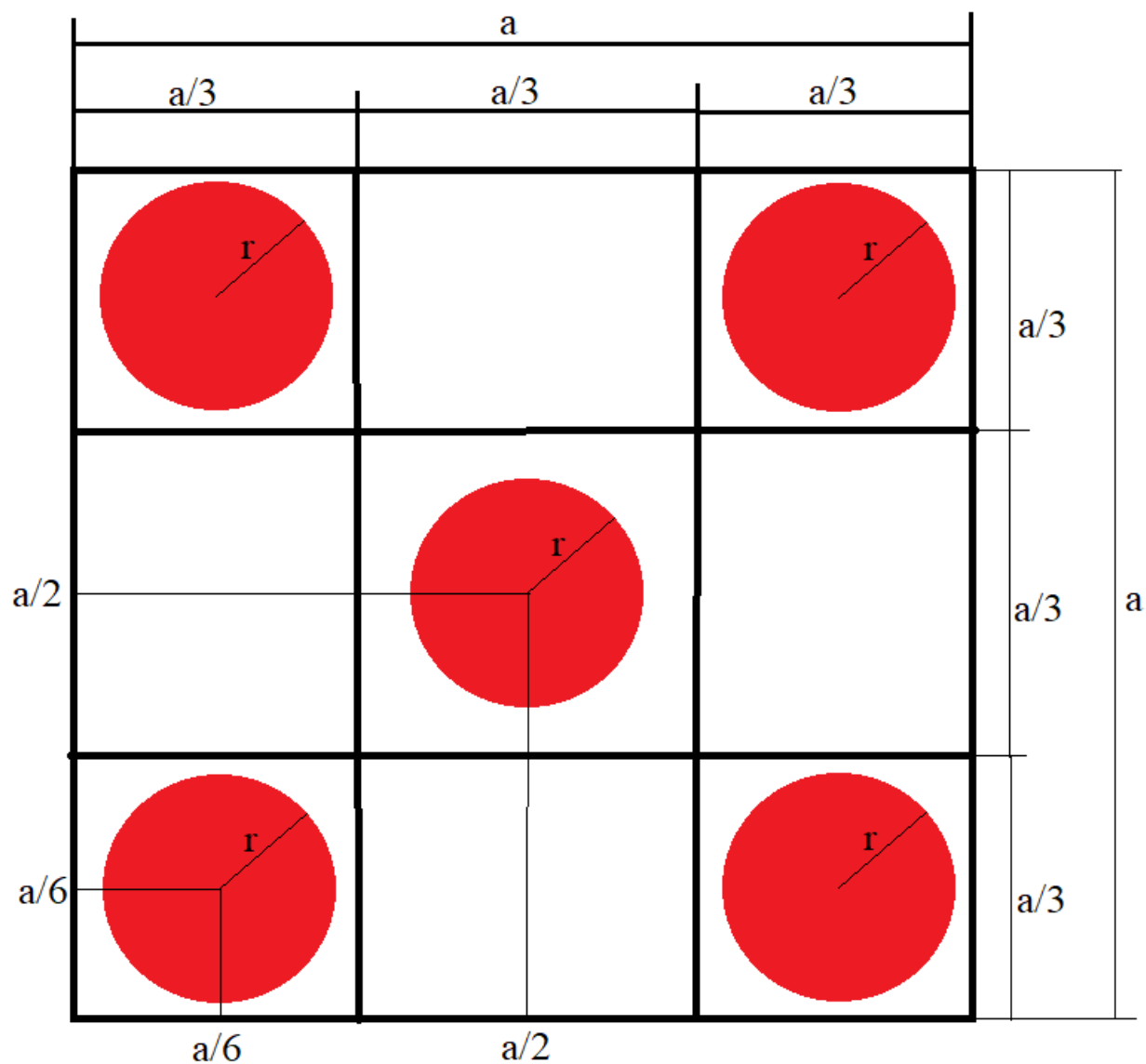


Рисунок 2.2 – Схема доски, вид сверху

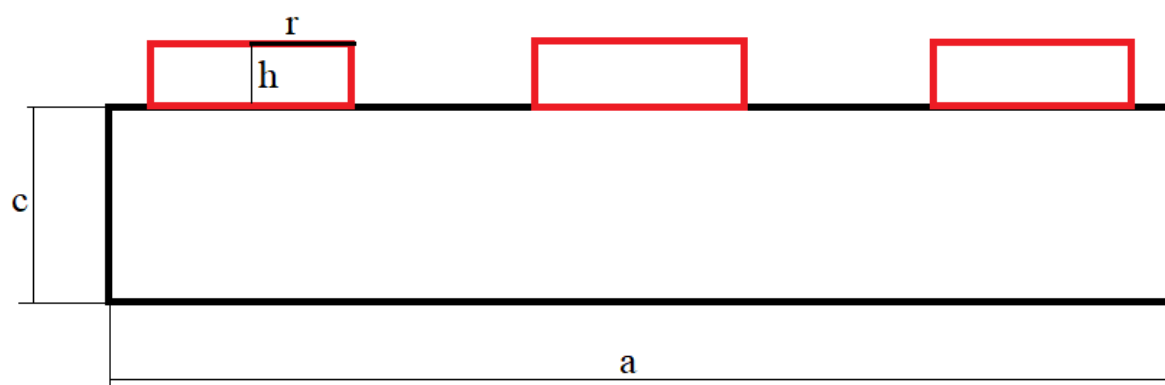


Рисунок 2.3 – Схема доски, вид сбоку

### 3 ОПИСАНИЕ АЛГОРИТМА

#### Основной алгоритм

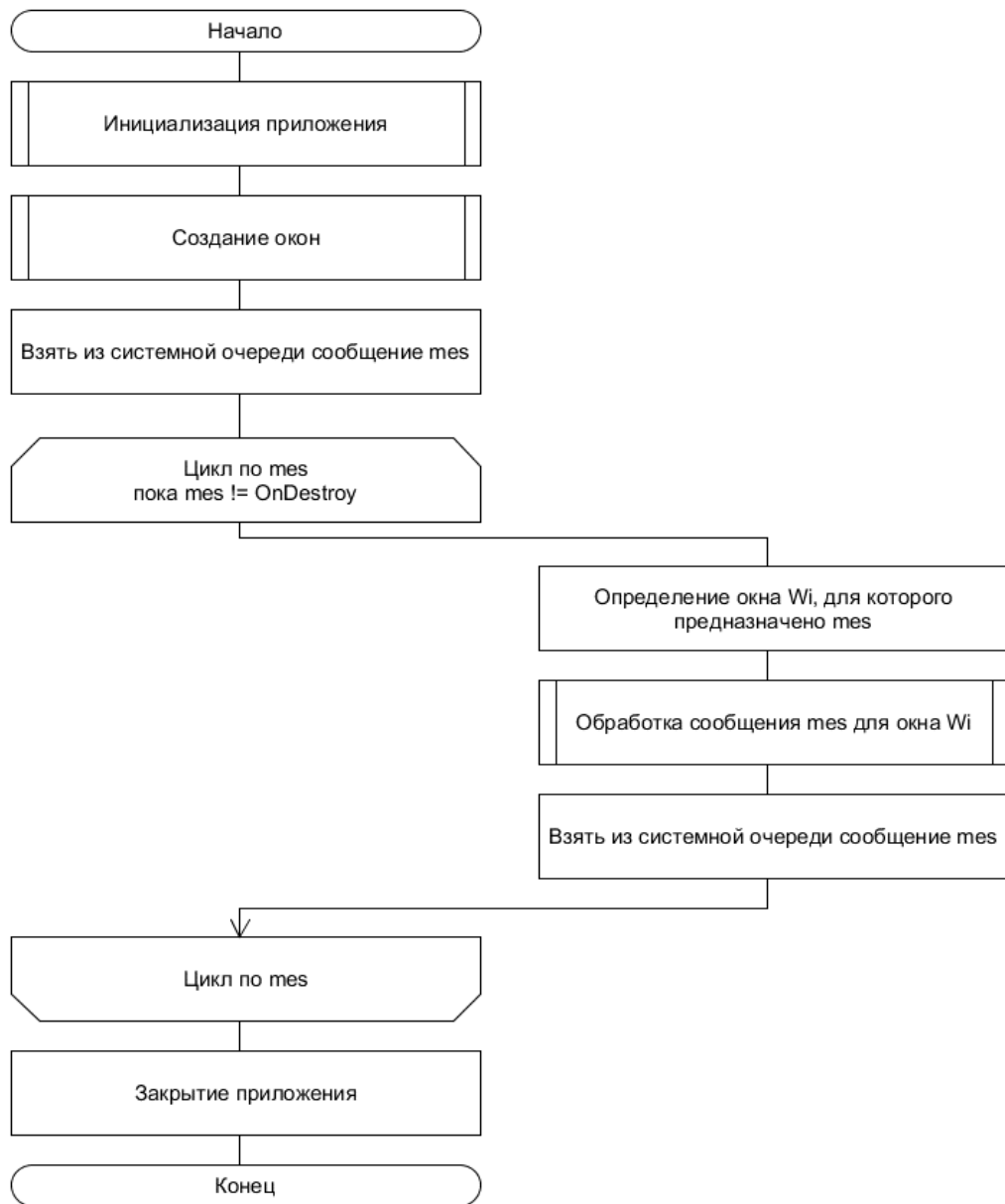


Рисунок 3.1 – Схема основного алгоритма

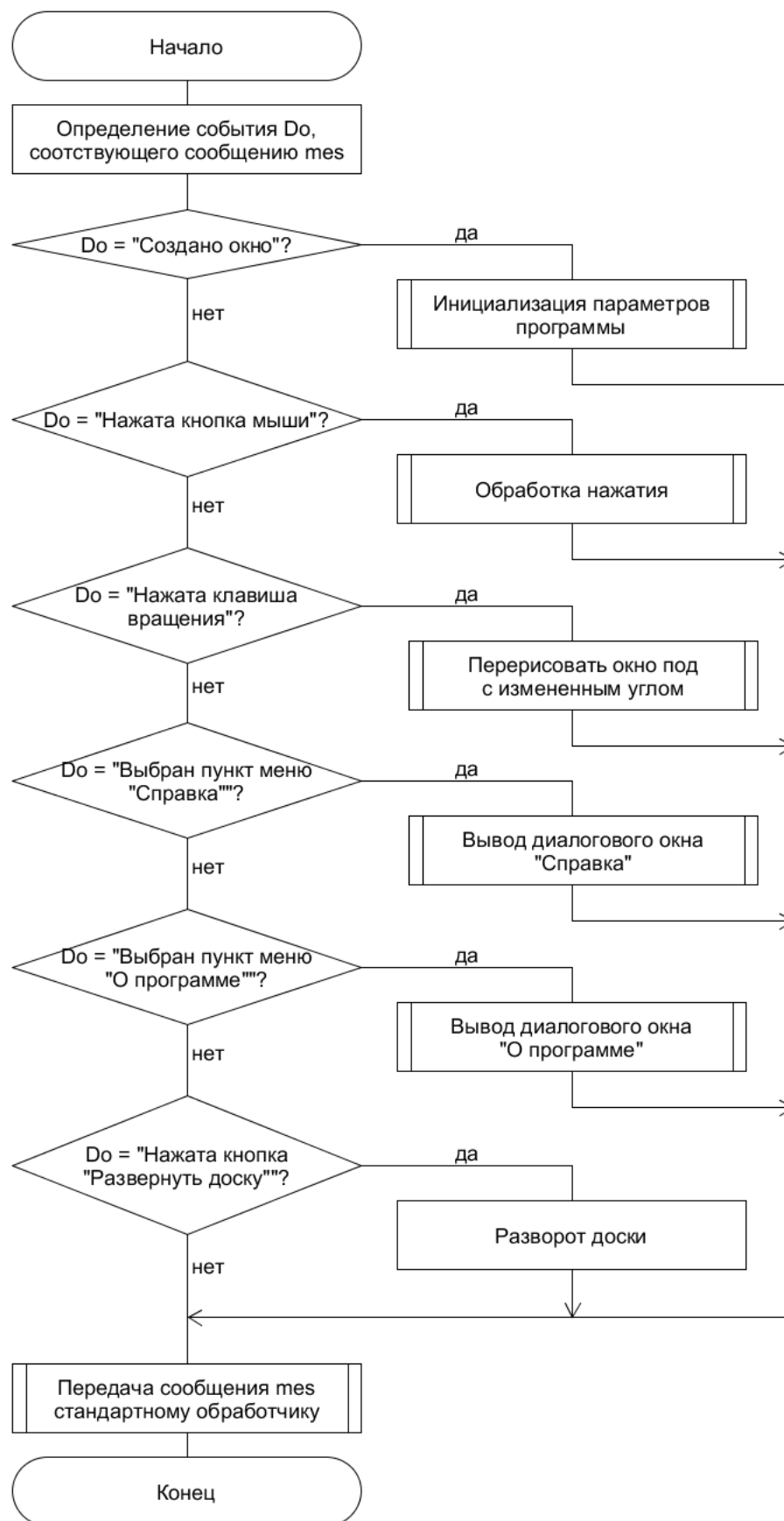


Рисунок 3.2 – Схема вспомогательного алгоритма «Обработка сообщения mes»

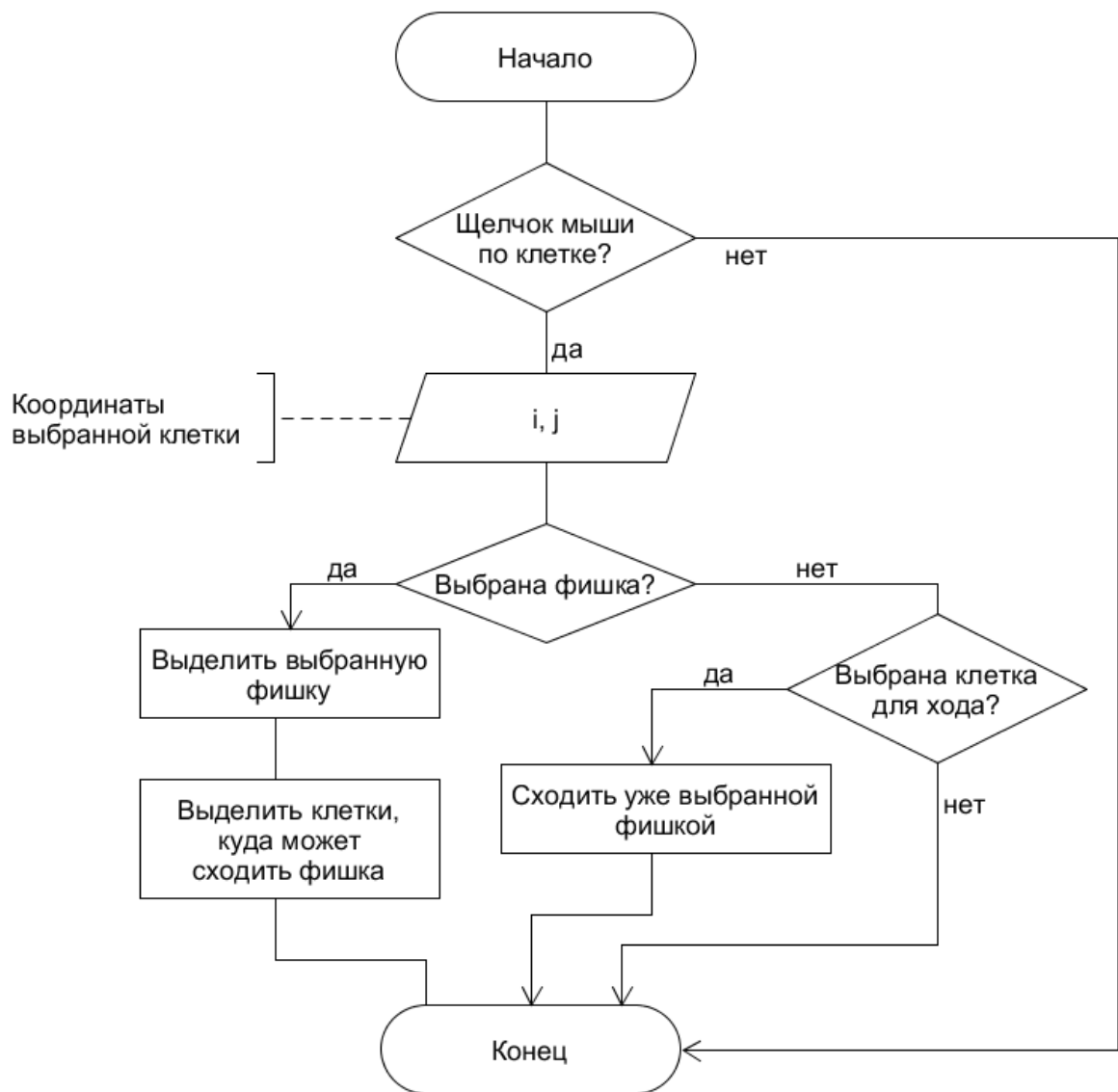


Рисунок 3.3 – Схема вспомогательного алгоритма «Обработка нажатия»

## 4 ОПИСАНИЕ ПРОГРАММЫ

В состав приложения входят 3 основных класса: класс Form1 отвечает за визуализацию формы, обработку событий, очередность ходов; класс Desk за визуализацию доски, вычисление координат выбранной клетки; класс Checkers берет на себя логику ходов шашек, их размещение и отрисовку.

На рисунке 4.1 представлена диаграмма классов, которая показывает, как они связаны. Пунктирная стрелка означает отношение использования.

Поля, методы классов можно посмотреть подробнее в приложении 1.

Для рисования в 3d используется компонент OpenTK.GLControl. (Рисунок 4.2)

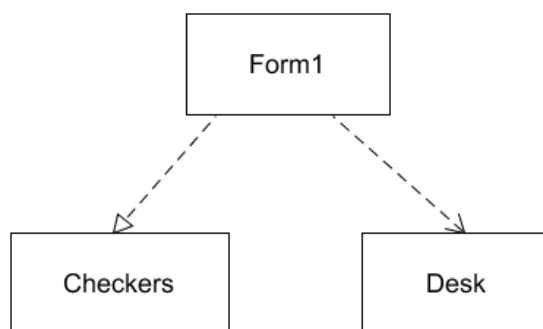


Рисунок 4.1 – Диаграмма классов

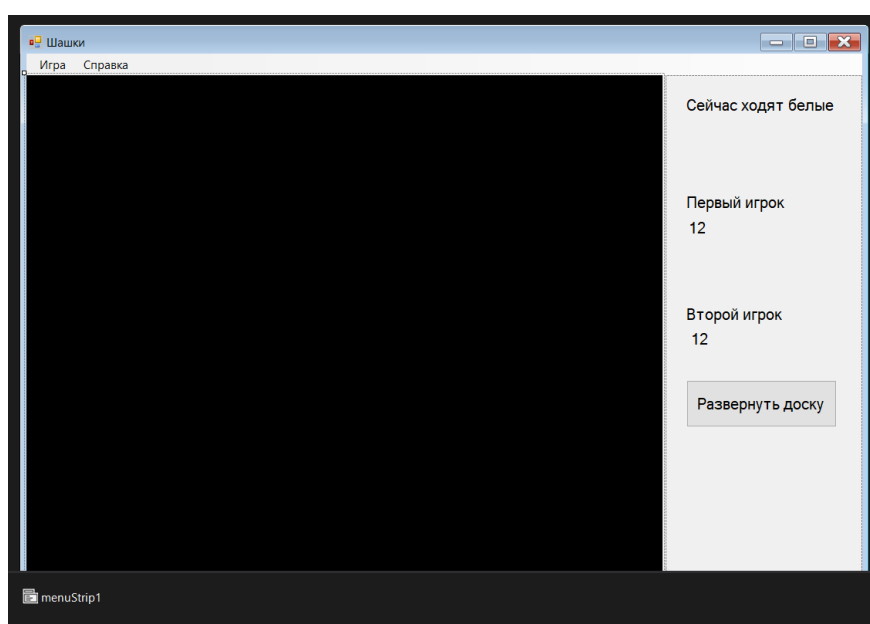


Рисунок 4.2 – Дизайн главной формы

## ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы были выбраны оптимальные графические компоненты и успешно внедрены в алгоритм решения. Разработанный код был проверен на контрольных тестах и в код были внесены необходимые исправления. Для приложения была разработана документация, описывающая ее установку и использование. Таким образом, цель работы была достигнута, задачи – решены.

Результаты работы можно использовать в процессе последующего обучения в форме навыков практического применения компьютерной графики для разработки сложных программных систем, понимания порядка этапов разработки программного обеспечения и достигаемых на каждом этапе результатов.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Аммерал, Л. Принципы программирования в машинной графике / Л. Аммерал. – М.: «Сол Систем», 1992. – 224 с.
2. Керниган, Б. Практика программирования / Б. Керниган, Р. Пайк. – М.: Вильямс, 2004. – 288 с.
3. Ласло, М. Вычислительная геометрия и компьютерная графика на C++ / М. Ласло. – М.: Бином, 1997. – 304 с.
4. Никулин, Е.А. Компьютерная графика. Модели и алгоритмы / Е.А. Никулин. – СПб.: Лань, 2018. – 708 с.
5. Порев, В.Н. Компьютерная графика / В.Н. Порев. – СПб.: БХВ-Петербург, 2005. – 428 с.

## ПРИЛОЖЕНИЕ 1 Текст программы

### Файл Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Imaging;
using System.Linq;
using System.Text;
using System.Windows.Forms;

using OpenTK;
using OpenTK.Graphics.OpenGL;
using OpenTK.Audio;
using OpenTK.Audio.OpenAL;
using OpenTK.Input;

namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        // углы вращения
        private float xrot = -40f;
        private float yrot = 0;
        private float zrot = 0;

        private const float MIN_ZOOM = 2.0f;
        private const float MAX_ZOOM = 15.0f;
        private float zoom = 3.0f; // текущее приближение
        private const float vZoom = 0.25f; // скорость
изменения зума

        // ближняя и дальняя плоскости отсечения
        private const float zNear = 1.0f;
        private const float zFar = 100.0f;

        private const float phi = 45.0f; // угол перспективной
проекции

        // скорость вращения
        private float dAngle = 1.5f;

        private const int nTextures = 3;
        private int[] textures = new int[nTextures];

        // размер доски
        private const int N = 8;
```



```

        // координаты выбранной клетки доски
        private Point chooseCell = new Point(-1, -1);

        // true - если сейчас ход белых,
        // false - если сейчас ход черных
        private bool isWhiteMove = true;

        private Desk desk;
        private Checkers figures;

        public Form1()
        {
            InitializeComponent();

            glControl1.MouseWheel += new
            MouseEventArgs(GlControl1_MouseWheel);
        }

        private void LoadTexture(Bitmap bmp)
        {
            BitmapData data = bmp.LockBits(
                new Rectangle(0, 0, bmp.Width, bmp.Height),
                ImageLockMode.ReadOnly,
                System.Drawing.Imaging.PixelFormat.Format24bppRgb);
            GL TexImage2D(TextureTarget.Texture2D, 0,
                PixelInternalFormat.Rgb, data.Width,
                data.Height, 0,
                OpenTK.Graphics.OpenGL.PixelFormat.Bgr,
                PixelType.UnsignedByte, data.Scan0);
            bmp.UnlockBits(data);
        }

        private void LoadTexture()
        {
            GL.GenTextures(nTextures, textures);

            Bitmap map = new Bitmap("images\\tree.bmp");
            GL.BindTexture(TextureTarget.Texture2D,
            textures[0]);
            GL TexParameter(TextureTarget.Texture2D,
            TextureParameterName.TextureMinFilter,
                (int)TextureMinFilter.Linear);
            GL TexParameter(TextureTarget.Texture2D,
            TextureParameterName.TextureMagFilter,
                (int)TextureMagFilter.Linear);
            LoadTexture(map);
            map = new Bitmap("images\\whiteCrown.bmp");
            GL.BindTexture(TextureTarget.Texture2D,
            textures[1]);
            GL TexParameter(TextureTarget.Texture2D,
            TextureParameterName.TextureMinFilter,
                (int)TextureMinFilter.Linear);

```

```

        GL TexParameter(TextureTarget.Texture2D,
TextureParameterName.TextureMagFilter,
            (int)TextureMagFilter.Linear);
        GL TexParameter(TextureTarget.Texture2D,
TextureParameterName.TextureWrapS,
            (int)TextureWrapMode.ClampToBorder);
        GL TexParameter(TextureTarget.Texture2D,
TextureParameterName.TextureWrapT,
            (int)TextureWrapMode.ClampToBorder);
        float[] borderColor = new float[3] { 0.0f, 0.0f,
0.0f };

        GL TexParameter(TextureTarget.Texture2D,
TextureParameterName.TextureBorderColor,
            borderColor);
        LoadTexture(map);

        map = new Bitmap("images\\blackCrown.bmp");
        GL.BindTexture(TextureTarget.Texture2D,
textures[2]);
        GL TexParameter(TextureTarget.Texture2D,
TextureParameterName.TextureMinFilter,
            (int)TextureMinFilter.Linear);
        GL TexParameter(TextureTarget.Texture2D,
TextureParameterName.TextureMagFilter,
            (int)TextureMagFilter.Linear);
        GL TexParameter(TextureTarget.Texture2D,
TextureParameterName.TextureWrapS,
            (int)TextureWrapMode.ClampToBorder);
        GL TexParameter(TextureTarget.Texture2D,
TextureParameterName.TextureWrapT,
            (int)TextureWrapMode.ClampToBorder);
        GL TexParameter(TextureTarget.Texture2D,
TextureParameterName.TextureBorderColor,
            borderColor);
        LoadTexture(map);
    }

    private void glControl1_Load(object sender, EventArgs
e)
    {

        GL.Color3(1.0f, 1.0f, 1.0f);
        GL.ClearColor(140.0f / 255, 203f / 255, 0.0f,
203.0f / 255);

        GL.ClearDepth(1.0);
        GL.DepthFunc(DepthFunction.Less);
        GL.Enable(EnableCap.DepthTest);

        GL.MatrixMode(MatrixMode.Projection);
        GL.LoadIdentity();
    }

```

```

        Matrix4 projection =
Matrix4.CreatePerspectiveFieldOfView(
            MathHelper.DegreesToRadians(phi),
            glControl1.AspectRatio, zNear, zFar);
GL.LoadMatrix(ref projection);
GL.MatrixMode(MatrixMode.Modelview);

LoadTexture();

GL.Enable(EnableCap.Lighting);
GL.Enable(EnableCap.Light0);
GL.Enable(EnableCap.ColorMaterial);
GL.Enable(EnableCap.Normalize);

desk = new Desk(1.0f, 1.0f, 0.125f, N,
textures[0]);
figures = new Checkers(N, 0.08, 1.0, 1.0, 0.125,
new float[3] { 1.0f, 1.0f, 1.0f },
    new float[3] { 1.0f, 0.0f, 0.0f }, new int[2]
{ textures[1], textures[2] });
    }

    private void GlControl1_MouseWheel(object sender,
System.Windows.Forms.MouseEventArgs e)
    {
        float delt = e.Delta * 1.0f /
SystemInformation.MouseWheelScrollDelta * vZoom;
        if (zoom + delt < MAX_ZOOM && zoom + delt >
MIN_ZOOM)
        {
            zoom += delt;
            glControl1.Invalidate();
        }
    }

    private void glControl1_Resize(object sender, EventArgs
e)
    {
        if (glControl1.Height == 0)
            glControl1.Height = 1;
        GL.Viewport(0, 0, glControl1.Width,
glControl1.Height);
        GL.MatrixMode(MatrixMode.Projection);
        GL.LoadIdentity();
        Matrix4 projection =
Matrix4.CreatePerspectiveFieldOfView(
            MathHelper.DegreesToRadians(phi),
            glControl1.AspectRatio, zNear, zFar);
        GL.LoadMatrix(ref projection);
        GL.MatrixMode(MatrixMode.Modelview);
    }

```

```

        private void glControl1_Paint(object sender,
PaintEventArgs e)
        {
            GL.Clear(ClearBufferMask.ColorBufferBit |
ClearBufferMask.DepthBufferBit);
            GL.LoadIdentity();
            GL.Translate(0.0, 0.0, -zoom);
            GL.Rotate(xrot, 1.0, 0.0, 0.0);
            GL.Rotate(yrot, 0.0, 1.0, 0.0);
            GL.Rotate(zrot, 0.0, 0.0, 1.0);

            desk.Draw(figures.Moves);
            figures.Draw(zrot > 90 && zrot < 270);

            GL.Flush();
            GL.Finish();

            glControl1.SwapBuffers();
        }

        private void glControl1_KeyDown(object sender,
KeyEventArgs e)
        {
            switch (e.KeyCode)
            {
                case Keys.Escape:
                    Close();
                    break;
                case Keys.Down:
                    xrot += dAngle;
                    break;
                case Keys.Up:
                    xrot -= dAngle;
                    break;
                case Keys.Left:
                    yrot -= dAngle;
                    break;
                case Keys.Right:
                    yrot += dAngle;
                    break;
            }
            glControl1.Invalidate();
        }

        private void glControl1_MouseClick(object sender,
System.Windows.Forms.MouseEventArgs e)
        {
            if (chooseCell.X != -1)
            {
                int res =
figures.IsWhiteOrBlack(chooseCell.Y, chooseCell.X);

```

```

        if (figures.IsSelect() && res != -1 &&
            (res == 0 && isWhiteMove || res ==
1 && !isWhiteMove))
        {
            figures.ClickCell(chooseCell);
            glControl1.Invalidate();
        }
        else if (!figures.IsSelect())
        {
            int code =
figures.ClickCell(chooseCell);
            // если ход состоялся
            if (code >= 0)
            {
                if (code >= 1)
                {
                    if (isWhiteMove)
                    {
                        scoreSecondGamer.Text =
(int.Parse(scoreSecondGamer.Text) - 1).
ToString();
                        if (scoreSecondGamer.Text
== "0")
                        {

                            glControl1.Invalidate();

                            MessageBox.Show("Игрок " + nameFirstGamer.Text +
                                " выиграл", "Игра закончилась");
                                ResetGame();
                                ChangeNames();
                                return;
                            }
                        }
                    else
                    {
                        scoreFirstGamer.Text =
(int.Parse(scoreFirstGamer.Text) - 1).ToString();
                        if (scoreFirstGamer.Text
== "0")
                        {

                            glControl1.Invalidate();

                            MessageBox.Show("Игрок " + nameSecondGamer.Text +
                                " выиграл", "Игра закончилась");
                                ResetGame();
                                ChangeNames();
                                return;
                            }
                        }
                    }
                }
            }
        }
    }
}

```

```

        bool isMove =
figures.ExistMove(!isWhiteMove);
        if (!isMove)
        {
            glControl1.Invalidate();
            if (isWhiteMove)
            {
                MessageBox.Show("Игрок "
+ nameFirstGamer.Text +
                " выиграл. У игрока
" + nameSecondGamer.Text +
                " нет возможности
ходить.", "Игра закончилась");
            }
            else
            {
                MessageBox.Show("Игрок "
+ nameSecondGamer.Text +
                " выиграл. У игрока
" + nameFirstGamer.Text +
                " нет возможности
ходить.", "Игра закончилась");
            }
            ResetGame();
            ChangeNames();
            return;
        }
        // если code == 2, то есть
возможность рубить дальше
        if (code != 2)
        {
            isWhiteMove = !isWhiteMove;
            if (isWhiteMove)
            {
                whoMove.Text = "Сейчас
ходят белые";
            }
            else
            {
                whoMove.Text = "Сейчас
ходят черные";
            }
        }
        glControl1.Invalidate();
    }
}
}

```

```

        private void glControl1_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
        {
            char key = e.KeyChar;
            if (key == 'd' || key == 'D' || key == 'B' || key
== 'B')
            {
                zrot += dAngle;
            }
            if (key == 'a' || key == 'A' || key == 'Ф' || key
== 'Ф')
            {
                zrot -= dAngle;
            }
            if (zrot < 0 || zrot > 360)
            {
                zrot = (int)zrot % 360;
                if (zrot < 0)
                {
                    zrot += 360;
                }
            }
            glControl1.Invalidate();
        }

        private void glControl1_PreviewKeyDown(object sender,
PreviewKeyDownEventArgs e)
        {
            switch (e.KeyCode)
            {
                case Keys.Down:
                case Keys.Up:
                case Keys.Left:
                case Keys.Right:
                    e.IsInputKey = true;
                    break;
            }
        }

        private void glControl1_MouseMove(object sender,
System.Windows.Forms.MouseEventArgs e)
        {
            desk.SelectCell(e.X, e.Y, out chooseCell);
            glControl1.Invalidate();
        }

        private void ChangeNames()
        {
            string t = nameFirstGamer.Text;
            nameFirstGamer.Text = nameSecondGamer.Text;
            nameSecondGamer.Text = t;
            MessageBox.Show("Игроки меняются цветами шашек",
"Сообщение");
        }

```

```

        private void ResetGame()
        {
            figures.Reset();
            scoreFirstGamer.Text = scoreSecondGamer.Text =
"12";
            whoMove.Text = "Сейчас ходят белые";
            isWhiteMove = true;
        }
        private void новаяИграToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            ResetGame();
        }

        private void оПрограммеToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            MessageBox.Show("Чумаков Денис, ЕТ-412" +
Environment.NewLine + "2021", "О программе");
        }
        private void правилаИгрыToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            System.Diagnostics.Process.Start("rules.html");
        }
        private void изменитьИменаToolStripMenuItem_Click(object
sender, EventArgs e)
        {
            Form2 form2 = new Form2();
            form2.NameFirstGamer = nameFirstGamer.Text;
            form2.NameSecondGamer = nameSecondGamer.Text;
            DialogResult res = form2.ShowDialog();
            if (res == DialogResult.OK)
            {
                nameFirstGamer.Text = form2.NameFirstGamer;
                nameSecondGamer.Text = form2.NameSecondGamer;
            }
        }
        private void reverseDeskButton_Click(object sender,
EventArgs e)
        {
            zrot = ((int)zrot - 180) % 360;
            if (zrot < 0)
            {
                zrot += 360;
            }
            glControl1.Focus();
            glControl1.Invalidate();
        }
    }
}

```



## Файл Desk.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;

using OpenTK;
//using OpenTK.Graphics;
using OpenTK.Graphics.OpenGL;
using OpenTK.Audio;
using OpenTK.Audio.OpenAL;
using OpenTK.Input;

namespace WindowsFormsApp1
{
    public class Desk
    {
        private readonly float a;
        private readonly float b;
        private readonly float c;
        private readonly float N;
        private readonly int iTexture;
        private Point chooseCell;

        public Desk(float a, float b, float c, int N, int
iTexture)
        {
            this.a = a;
            this.b = b;
            this.c = c;
            this.N = N;
            this.iTexture = iTexture;
            chooseCell = new Point(-1, -1);
        }

        public void Draw(Checkers.KindMove[,] moves)
        {
            // Рисуем 6 граней
            // Передняя грань
            for (int i = 0; i < N; ++i)
            {
                for (int j = 0; j < N; ++j)
                {
                    if ((i + j) % 2 != 0)
                    {
                        GL.Color3(1.0f, 1.0f, 1.0f); //
белая клетка
                    }
                }
            }
        }
    }
}
```

```

else
{
    GL.Color3(0.0f, 0.0f, 0.0f); //
черная клетка

    if (moves[i, j] ==
Checkers.KindMove.Yes)
    {
        GL.Color3(1.0f, 1.0f, 0.0f); //
желтая клетка
    }

    if (chooseCell.X == j && chooseCell.Y ==
i)
    {
        GL.Color3(0.0f, 0.0f, 1.0f);
// синяя клетка
    }

    float x = -a + 2 * a / N * j;
    float y = -b + 2 * b / N * i;
    float dx = 2 * a / N;
    float dy = 2 * b / N;
    GL.Begin(PrimitiveType.Quads);

    GL.Normal3(0.0f, 0.0f, 1.0f);

    GL.Vertex3(x, y, c); // Нижняя левая
точка

    GL.Vertex3(x + dx, y, c); // Нижняя
правая точка

    GL.Vertex3(x + dx, y + dy, c); //
Верхняя правая точка

    GL.Vertex3(x, y + dy, c); // Верхняя
левая точка

    GL.End();
}

GL.Enable(EnableCap.Texture2D);
GL.Color3(Color.Transparent);
// Задняя грань
GL.BindTexture(TextureTarget.Texture2D, iTexture);
GL.Begin(PrimitiveType.Quads);

GL.Normal3(0.0f, 0.0f, -1.0f);

GL.TexCoord2(1.0f, 0.0f);
GL.Vertex3(-a, -b, -c); // Нижняя правая точка

```

```

GL.TexCoord2(1.0f, 1.0f);
GL.Vertex3(-a, b, -c); // Верхняя правая точка

GL.TexCoord2(0.0f, 1.0f);
GL.Vertex3(a, b, -c); // Верхняя левая точка

GL.TexCoord2(0.0f, 0.0f);
GL.Vertex3(a, -b, -c); // Нижняя левая точка
GL.End();

// Верхняя грань
GL.BindTexture(TextureTarget.Texture2D, iTexture);
GL.Begin(PrimitiveType.Quads);

GL.Normal3(0.0f, 1.0f, 0.0f);

GL.TexCoord2(0.0f, 1.0f);
GL.Vertex3(-a, b, -c); // Верхняя левая точка

GL.TexCoord2(1.0f, 1.0f);
GL.Vertex3(-a, b, c); // Нижняя левая точка

GL.TexCoord2(1.0f, 0.0f);
GL.Vertex3(a, b, c); // Нижняя правая точка

GL.TexCoord2(0.0f, 0.0f);
GL.Vertex3(a, b, -c); // Верхняя правая точка
GL.End();

// Нижняя грань
GL.BindTexture(TextureTarget.Texture2D, iTexture);
GL.Begin(PrimitiveType.Quads);

GL.Normal3(0.0f, -1.0f, 0.0f);

GL.TexCoord2(1.0f, 0.0f);
GL.Vertex3(-a, -b, -c); // Верхняя правая точка

GL.TexCoord2(1.0f, 1.0f);
GL.Vertex3(a, -b, -c); // Верхняя левая точка

GL.TexCoord2(0.0f, 1.0f);
GL.Vertex3(a, -b, c); // Нижняя левая точка

GL.TexCoord2(0.0f, 0.0f);
GL.Vertex3(-a, -b, c); // Нижняя правая точка
GL.End();

// Правая грань
GL.BindTexture(TextureTarget.Texture2D, iTexture);
GL.Begin(PrimitiveType.Quads);

```

```

        GL.Normal3(1.0f, 0.0f, 0.0f);

        GL.TexCoord2(1.0f, 0.0f);
        GL.Vertex3(a, -b, -c); // Нижняя правая точка

        GL.TexCoord2(1.0f, 1.0f);
        GL.Vertex3(a, b, -c); // Верхняя правая точка

        GL.TexCoord2(0.0f, 1.0f);
        GL.Vertex3(a, b, c); // Верхняя левая точка

        GL.TexCoord2(0.0f, 0.0f);
        GL.Vertex3(a, -b, c); // Нижняя левая точка
        GL.End();

        // Левая грань
        GL.BindTexture(TextureTarget.Texture2D, iTexture);
        GL.Begin(PrimitiveType.Quads);

        GL.Normal3(-1.0f, 0.0f, 0.0f);

        GL.TexCoord2(0.0f, 0.0f);
        GL.Vertex3(-a, -b, -c); // Нижняя левая точка

        GL.TexCoord2(1.0f, 0.0f);
        GL.Vertex3(-a, -b, c); // Нижняя правая точка

        GL.TexCoord2(1.0f, 1.0f);
        GL.Vertex3(-a, b, c); // Верхняя правая точка

        GL.TexCoord2(0.0f, 1.0f);
        GL.Vertex3(-a, b, -c); // Верхняя левая точка
        GL.End(); // Done Drawing The Cube

        GL.Disable(EnableCap.Texture2D);
    }
    static private double[] GetSpaceCoord(int winX, int
winY)
    {
        int[] vp = new int[4];
        double[] mMatrix = new double[16],
            pMatrix = new double[16];
        float zt = 0.0f;

        GL.GetInteger(GetPName.Viewport, vp);
        winY = vp[3] - winY;

        GL.GetDouble(GetPName.ModelviewMatrix, mMatrix);
        GL.GetDouble(GetPName.ProjectionMatrix, pMatrix);
        GL.ReadPixels(winX, winY, 1, 1,
OpenTK.Graphics.OpenGL.PixelFormat.DepthComponent,
            PixelType.Float, ref zt);
    }

```

```

        double x, y, z;
        Tao.OpenGl.Glu.gluUnProject(winX, winY, zt,
mMatrix, pMatrix, vp, out x, out y, out z);
        double[] result = new double[3] { x, y, z };
        return result;
    }

    public void SelectCell(int winX, int winY, out Point p)
    {
        double[] spaceCoord = GetSpaceCoord(winX, winY);
        double x = spaceCoord[0];
        double y = spaceCoord[1];
        for (int i = 0; i < N; ++i)
        {
            for (int j = 0; j < N; ++j)
            {
                if ((i + j) % 2 == 0)
                {
                    double x1 = -a + 2 * a / N * j;
                    double y1 = -b + 2 * b / N * i;
                    double dx = 2 * a / N;
                    double dy = 2 * b / N;
                    if (x1 < x && x < x1 + dx && y1 < y
&& y < y1 + dy)
                    {
                        chooseCell.Y = i;
                        chooseCell.X = j;
                        p = new Point(chooseCell.X,
chooseCell.Y);
                        return;
                    }
                }
            }
        }
        chooseCell.Y = -1;
        chooseCell.X = -1;
        p = new Point(-1, -1);
    }
}

```

## Файл Checkers.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Drawing;
using OpenTK;
//using OpenTK.Graphics;
using OpenTK.Graphics.OpenGL;
using OpenTK.Audio;
using OpenTK.Audio.OpenAL;
using OpenTK.Input;

namespace WindowsFormsApp1
{
    public class Checkers
    {
        private readonly int N; // размер доски NxN
        private readonly float[] colorWhite; // цвет белых шашек
        private readonly float[] colorBlack; // цвет черных
        шашек
        private readonly int n; // количество точек цилиндра
        private readonly int[] textures; // текстуры
        private readonly double r; // радиус цилиндра
        private readonly double h; // высота цилиндра
        private readonly double a; // сторона доски по оси Oх
        длиной 2a
        private readonly double b; // сторона доски по оси Oу
        длиной 2b
        private readonly double c; // сторона доски по оси Oz
        длиной 2c
        public enum KindMove { No, Yes };

        // виды шашек
        private enum KindCell { Free, White, Black, WhiteQueen,
        BlackQueen};
        private KindCell[,] figure; // матрица расположения
        шашек
        private KindMove[,] moves; // матрица возможных ходов
        private Point chooseFigure; // выбранная шашка

        public Checkers(int N, double h, double a, double b,
        double c,
            float[] colorWhite, float[] colorBlack, int[]
        textures)
        {
            this.N = N;
            this.colorWhite = colorWhite;
            this.colorBlack = colorBlack;
            this.n = 20;
            this.textures = textures;
        }
    }
}
```

```

this.r = a / N - 0.025;
this.h = h;
this.a = a;
this.b = b;
this.c = c;
chooseFigure = new Point(-1, -1);

moves = new KindMove[N, N];
figure = new KindCell[N, N];
for (int i = 0; i < N; ++i)
{
    for (int j = 0; j < N; ++j)
    {
        moves[i, j] = KindMove.No;

        figure[i, j] = KindCell.Free;
        if ((i + j) % 2 == 0)
        {
            if (i < 3)
            {
                figure[i, j] = KindCell.White;
            }
            else if (i >= N - 3)
            {
                figure[i, j] = KindCell.Black;
            }
        }
    }
}
//figure[5, 5] = KindCell.BlackQueen;
//figure[2, 4] = KindCell.WhiteQueen;
//figure[7, 5] = KindCell.Black;
}

// задать начальное расположение шашек
public void Reset()
{
    chooseFigure = new Point(-1, -1);
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
        {
            moves[i, j] = KindMove.No;

            figure[i, j] = KindCell.Free;
            if ((i + j) % 2 == 0)
            {
                if (i < 3)
                {
                    figure[i, j] = KindCell.White;
                }
            }
        }
    }
}

```

```

        else if (i >= N - 3)
        {
            figure[i, j] = KindCell.Black;
        }
    }
}

}

}

public KindMove[,] Moves { get { return moves; } }

// нарисовать шашку в виде цилиндра
private void DrawCylinder(int i0, int j0, KindCell kind,
bool isInverseCrown)
{
    double eps = h / 10;
    double dx = -a + a / N + 2.0 / N * j0;
    double dy = -b + b / N + 2.0 / N * i0;
    float[] color;
    if (kind == KindCell.White || kind ==
KindCell.WhiteQueen)
    {
        color = colorWhite;
    }
    else
    {
        color = colorBlack;
    }
    GL.Color3(color);

    // боковая поверхность
    GL.Begin(BeginMode.QuadStrip);
    for (int i = 0; i <= n; ++i)
    {
        double angle = 2 * Math.PI / n * i;
        double x = r * Math.Cos(angle) + dx;
        double y = r * Math.Sin(angle) + dy;

        GL.Normal3(x - dx, y - dy, 0.0f);

        GL.Vertex3(x, y, c);
        GL.Vertex3(x, y, c + h - eps);
    }
    GL.End();

    GL.Color3(Color.Black);
    GL.Begin(BeginMode.QuadStrip);
    for (int i = 0; i <= n; ++i)
    {
        double angle = 2 * Math.PI / n * i;
        double x = r * Math.Cos(angle) + dx;
        double y = r * Math.Sin(angle) + dy;

```



```

        GL.Normal3(0.0f, 0.0f, 1.0f);

        GL.Vertex3(x, y, c + h - eps);
        GL.Vertex3(x, y, c + h);
    }
    GL.End();

    GL.Color3(color);
    if (kind == KindCell.WhiteQueen || kind ==
KindCell.BlackQueen)
    {
        GL.Enable(EnableCap.Texture2D);
        GL.Color3(Color.Transparent);
        if (kind == KindCell.WhiteQueen)
        {
            GL.BindTexture(TextureTarget.Texture2D,
textures[0]);
        }
        else
        {
            GL.BindTexture(TextureTarget.Texture2D,
textures[1]);
        }
    }

    // основание цилиндра
    GL.Begin(BeginMode.TriangleFan);
    GL.Normal3(0.0f, 0.0f, 1.0f);
    GL.TexCoord2(0.5f, 0.5f);
    GL.Vertex3(dx, dy, c + h);
    for (int i = 0; i <= n; ++i)
    {
        double angle = 2 * Math.PI / n * i;
        double x = r * Math.Cos(angle) + dx;
        double y = r * Math.Sin(angle) + dy;

        int sign = isInverseCrown ? 1 : -1;

        GL.TexCoord2((Math.Cos(angle) + 1) / 2, (sign *
Math.Sin(angle) + 1) / 2);
        GL.Vertex3(x, y, c + h);
    }
    GL.End();
    if (kind == KindCell.WhiteQueen || kind ==
KindCell.BlackQueen)
    {
        GL.Disable(EnableCap.Texture2D);
    }
}

```

```

public void Draw(bool isInverseQueen)
{
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
        {
            if ((i + j) % 2 == 0 && figure[i, j] !=
KindCell.Free)
            {
                DrawCylinder(i, j, figure[i, j],
isInverseQueen);
            }
        }
    }
}

private void ClearMoves()
{
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
        {
            moves[i, j] = KindMove.No;
        }
    }
}

public bool IsSelect()
{
    return chooseFigure.X == -1;
}

public int IsWhiteOrBlack(int i, int j)
{
    switch (figure[i, j])
    {
        case KindCell.White:
        case KindCell.WhiteQueen:
            return 0;
        case KindCell.Black:
        case KindCell.BlackQueen:
            return 1;
        default:
            return -1;
    }
}

```

```

private bool TestCutQueen(KindCell test1, KindCell
test2, int a, int b,
    bool isDo, ref int j0, ref bool isCut)
{
    // если пока еще не встретили вражескую шашку
    if (j0 == -1)
    {
        // все же встретили вражескую шашку
        if (figure[a, b] == test1 || figure[a, b] ==
test2)
        {
            // запоминаем ее положение
            j0 = b;
        }
        // иначе если встретили свою, выходим
        else if (figure[a, b] != KindCell.Free)
        {
            return false;
        }
    }
    // если до этого встречали вражескую шашку и
находимся на свободной,
    // пометим ее
    else if (figure[a, b] == KindCell.Free)
    {
        if (isDo)
        {
            moves[a, b] = KindMove.Yes;
        }
        isCut = true;
    }
    // если до этого встречали вражескую шашку и
встретили другую шашку,
    // выходим
    else
    {
        return false;
    }
    return true;
}

private bool SimpleMoveQueen(int i, int j, bool isDo)
{
    bool isMove = false;
    for (int b = j + 1; b < N; ++b)
    {
        int a = b + i - j;
        if (a >= 0 && a < N)
        {
            if (figure[a, b] == KindCell.Free)
            {
                isMove = true;
            }
        }
    }
}

```

```

        if (isDo)
        {
            moves[a, b] = KindMove.Yes;
        }
    }
    else
    {
        break;
    }
}

for (int b = j + 1; b < N; ++b)
{
    int a = -b + i + j;
    if (a >= 0 && a < N)
    {
        if (figure[a, b] == KindCell.Free)
        {
            isMove = true;
            if (isDo)
            {
                moves[a, b] = KindMove.Yes;
            }
        }
        else
        {
            break;
        }
    }
}

for (int b = j - 1; b >= 0; --b)
{
    int a = b + i - j;
    if (a >= 0 && a < N)
    {
        if (figure[a, b] == KindCell.Free)
        {
            isMove = true;
            if (isDo)
            {
                moves[a, b] = KindMove.Yes;
            }
        }
        else
        {
            break;
        }
    }
}

```

```

for (int b = j - 1; b >= 0; --b)
{
    int a = -b + i + j;
    if (a >= 0 && a < N)
    {
        if (figure[a, b] == KindCell.Free)
        {
            isMove = true;
            if (isDo)
            {
                moves[a, b] = KindMove.Yes;
            }
        }
        else
        {
            break;
        }
    }
}
return isMove;
}

private bool SimpleMove(int i, int j, bool isDo)
{
    bool isMove = false;
    if (figure[i, j] == KindCell.White && i + 1 < N && j
- 1 >= 0 &&
        figure[i + 1, j - 1] == KindCell.Free)
    {
        isMove = true;
        if (isDo)
        {
            moves[i + 1, j - 1] = KindMove.Yes;
        }
    }
    if (figure[i, j] == KindCell.White && i + 1 < N && j
+ 1 < N &&
        figure[i + 1, j + 1] == KindCell.Free)
    {
        isMove = true;
        if (isDo)
        {
            moves[i + 1, j + 1] = KindMove.Yes;
        }
    }

    if (figure[i, j] == KindCell.Black && i - 1 >= 0 &&
j - 1 >= 0 &&
        figure[i - 1, j - 1] == KindCell.Free)
    {
        isMove = true;
    }
}

```

```

        if (isDo)
        {
            moves[i - 1, j - 1] = KindMove.Yes;
        }
    }
    if (figure[i, j] == KindCell.Black && i - 1 >= 0 &&
j + 1 < N &&
        figure[i - 1, j + 1] == KindCell.Free)
    {
        isMove = true;
        if (isDo)
        {
            moves[i - 1, j + 1] = KindMove.Yes;
        }
    }
    return isMove;
}

// есть ли у шашки цвета kind (i,j) возможность рубить
private bool TestCut(KindCell kind, int i, int j, bool
isDo)
{
    KindCell test1, test2;
    bool isCut = false;
    if (kind == KindCell.White || kind ==
KindCell.WhiteQueen)
    {
        test1 = KindCell.Black;
        test2 = KindCell.BlackQueen;
    }
    else
    {
        test1 = KindCell.White;
        test2 = KindCell.WhiteQueen;
    }

    // отдельно обрабатываем дамки
    if (kind == KindCell.WhiteQueen || kind ==
KindCell.BlackQueen)
    {
        int j0 = -1;
        for (int b = j + 1; b < N; ++b)
        {
            int a = b + i - j;
            if (a >= 0 && a < N)
            {
                if (!TestCutQueen(test1, test2, a, b,
isDo, ref j0, ref isCut))
                {
                    break;
                }
            }
        }
    }
}

```

```

        j0 = -1;
        for (int b = j - 1; b >= 0; --b)
        {
            int a = b + i - j;
            if (a >= 0 && a < N)
            {
                if (!TestCutQueen(test1, test2, a, b,
isDo, ref j0, ref isCut))
                {
                    break;
                }
            }
        }

        j0 = -1;
        for (int b = j + 1; b < N; ++b)
        {
            int a = -b + i + j;
            if (a >= 0 && a < N)
            {
                if (!TestCutQueen(test1, test2, a, b,
isDo, ref j0, ref isCut))
                {
                    break;
                }
            }
        }

        j0 = -1;
        for (int b = j - 1; b >= 0; --b)
        {
            int a = -b + i + j;
            if (a >= 0 && a < N)
            {
                if (!TestCutQueen(test1, test2, a, b,
isDo, ref j0, ref isCut))
                {
                    break;
                }
            }
        }
        return isCut;
    }

```

```

        if (i + 1 < N && j - 1 >= 0 && (figure[i + 1, j - 1]
== test1 ||
        figure[i + 1, j - 1] == test2) &&
        i + 2 < N && j - 2 >= 0 && figure[i + 2, j -
2] == KindCell.Free)
        {
            if (isDo)
            {
                moves[i + 2, j - 2] = KindMove.Yes;
            }
            isCut = true;
        }

        if (i + 1 < N && j + 1 < N && (figure[i + 1, j + 1]
== test1 ||
        figure[i + 1, j + 1] == test2) &&
        i + 2 < N && j + 2 < N && figure[i + 2, j +
2] == KindCell.Free)
        {
            if (isDo)
            {
                moves[i + 2, j + 2] = KindMove.Yes;
            }
            isCut = true;
        }

        if (i - 1 >= 0 && j - 1 >= 0 && (figure[i - 1, j -
1] == test1 ||
        figure[i - 1, j - 1] == test2) &&
        i - 2 >= 0 && j - 2 >= 0 && figure[i - 2, j
- 2] == KindCell.Free)
        {
            if (isDo)
            {
                moves[i - 2, j - 2] = KindMove.Yes;
            }
            isCut = true;
        }

        if (i - 1 >= 0 && j + 1 < N && (figure[i - 1, j + 1]
== test1 ||
        figure[i - 1, j + 1] == test2) &&
        i - 2 >= 0 && j + 2 < N && figure[i - 2, j +
2] == KindCell.Free)
        {
            if (isDo)
            {
                moves[i - 2, j + 2] = KindMove.Yes;
            }
            isCut = true;
        }
        return isCut;
    }
}

```



```

// есть ли у какой-либо шашки цвета kind возможность
рубить
private bool TestAllCut(KindCell kind)
{
    KindCell partner = KindCell.White;
    switch (kind)
    {
        case KindCell.White:
            partner = KindCell.WhiteQueen;
            break;
        case KindCell.Black:
            partner = KindCell.BlackQueen;
            break;
        case KindCell.BlackQueen:
            partner = KindCell.Black;
            break;
    }
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
        {
            if ((i + j) % 2 == 0 && (figure[i, j] ==
kind ||
                                figure[i, j] == partner) &&
                                TestCut(figure[i, j], i, j, false))
            {
                return true;
            }
        }
    }
    return false;
}

private void IsCutMove(int i, int j, out Point p3)
{
    p3 = new Point(-1, -1);
    Point p1 = new Point(chooseFigure.Y,
chooseFigure.X);
    Point p2 = new Point(i, j);
    KindCell test1 = KindCell.Black;
    KindCell test2 = KindCell.BlackQueen;
    if (figure[p1.X, p1.Y] == KindCell.Black ||
        figure[p1.X, p1.Y] == KindCell.BlackQueen)
    {
        test1 = KindCell.White;
        test2 = KindCell.WhiteQueen;
    }
}

```

```

if (p1.X < p2.X && p1.Y < p2.Y)
{
    for (int b = p1.Y + 1; b < p2.Y; ++b)
    {
        int a = b + p1.X - p1.Y;
        if (a < 0 || a >= N)
        {
            continue;
        }
        if (figure[a, b] == test1 || figure[a, b] ==
test2)
        {
            p3 = new Point(a, b);
            break;
        }
        else if (figure[a, b] != KindCell.Free)
        {
            break;
        }
    }
}
else if (p1.X < p2.X && p1.Y > p2.Y)
{
    for (int b = p1.Y - 1; b > p2.Y; --b)
    {
        int a = -b + p1.Y + p1.X;
        if (a < 0 || a >= N)
        {
            continue;
        }
        if (figure[a, b] == test1 || figure[a, b] ==
test2)
        {
            p3 = new Point(a, b);
            break;
        }
        else if (figure[a, b] != KindCell.Free)
        {
            break;
        }
    }
}
else if (p1.X > p2.X && p1.Y < p2.Y)
{
    for (int b = p1.Y + 1; b < p2.Y; ++b)
    {
        int a = -b + p1.Y + p1.X;
        if (a < 0 || a >= N)
        {
            continue;
        }
    }
}

```

```

        if (figure[a, b] == test1 || figure[a, b] ==
test2)
        {
            p3 = new Point(a, b);
            break;
        }
        else if (figure[a, b] != KindCell.Free)
        {
            break;
        }
    }
}
else if (p1.X > p2.X && p1.Y > p2.Y)
{
    for (int b = p1.Y - 1; b > p2.Y; --b)
    {
        int a = b + p1.X - p1.Y;
        if (a < 0 || a >= N)
        {
            continue;
        }
        if (figure[a, b] == test1 || figure[a, b] ==
test2)
        {
            p3 = new Point(a, b);
            break;
        }
        else if (figure[a, b] != KindCell.Free)
        {
            break;
        }
    }
}

}

public int ClickCell(Point cell)
{
    int i = cell.Y;
    int j = cell.X;
    int count = 0;
    // шашка не была выбрана
    if (chooseFigure.X == -1)
    {
        // если шашка не была выбрана и выбрана пустая
клетка, то выходим
        if (figure[i, j] == KindCell.Free)
        {
            return -1;
        }

        bool isCut = TestCut(figure[i, j], i, j, true);

        // если шашка не рубит

```

```

        if (!isCut)
        {
            // если есть возможность рубить у других
            шашек, выходим
            if (TestAllCut(figure[i, j]))
            {
                return -1;
            }

            bool isMove = (figure[i, j] ==
            KindCell.WhiteQueen || figure[i, j] == KindCell.BlackQueen) ?
                SimpleMoveQueen(i, j, true) :
            SimpleMove(i, j, true);

            // если ходов нет, выходим
            if (!isMove)
            {
                return -1;
            }

            chooseFigure = new Point(j, i);
            moves[i, j] = KindMove.Yes;
        }
        // если шашка была выбрана
        else
        {
            // если шашка была выбрана и выбрана пустая
            клетка
            // не означающая ход, то выходим
            if (figure[i, j] == KindCell.Free && moves[i, j]
            == KindMove.No)
            {
                return -1;
            }
            // если щелкнули по уже выбранной шашке, тогда
            сбрасываем выбор и выходим
            if (i == chooseFigure.Y && j == chooseFigure.X)
            {
                ClearMoves();
                chooseFigure.Y = -1;
                chooseFigure.X = -1;
                return -1;
            }
            else if (IsWhiteOrBlack(i, j) ==
            IsWhiteOrBlack(chooseFigure.Y, chooseFigure.X))
            {
                ClearMoves();
                chooseFigure.Y = -1;
                chooseFigure.X = -1;
                ClickCell(new Point(j, i));
                return -1;
            }
        }
    }
}

```

```

        // на другие шашки не обращаем внимания
        else if (figure[i, j] != KindCell.Free)
        {
            return -1;
        }

        Point p3;
        IsCutMove(i, j, out p3);

        // сделаем ход
        figure[i, j] = figure[chooseFigure.Y,
chooseFigure.X];
        figure[chooseFigure.Y, chooseFigure.X] =
KindCell.Free;

        // белая пешка стала дамкой
        if (figure[i, j] == KindCell.White && i == N -
1)
        {
            figure[i, j] = KindCell.WhiteQueen;
        }

        // черная пешка стала дамкой
        if (figure[i, j] == KindCell.Black && i == 0)
        {
            figure[i, j] = KindCell.BlackQueen;
        }

        // рубим
        if (p3.X != -1)
        {
            figure[p3.X, p3.Y] = KindCell.Free;
            // если есть возможность рубить дальше,
сообщим об этом
            if (TestCut(figure[i, j], i, j, false))
            {
                ClearMoves();
                chooseFigure = new Point(-1, -1);
                ClickCell(new Point(j, i));
                count = 2;
                return count;
            }
            else
            {
                count = 1;
            }
        }

        ClearMoves();
        chooseFigure = new Point(-1, -1);
    }
    return count;
}

```

```

public bool ExistMove(bool isWhite)
{
    KindCell k1 = KindCell.White, k2 =
KindCell.WhiteQueen;
    if (!isWhite)
    {
        k1 = KindCell.Black;
        k2 = KindCell.BlackQueen;
    }
    if (TestAllCut(k1))
    {
        return true;
    }
    for (int i = 0; i < N; ++i)
    {
        for (int j = 0; j < N; ++j)
        {
            if ((i + j) % 2 == 0 && (figure[i, j] == k1
&& SimpleMove(i, j, false) ||
            figure[i, j] == k2 && SimpleMoveQueen(i,
j, false)))
            {
                return true;
            }
        }
    }
    return false;
}
}
}

```

## ПРИЛОЖЕНИЕ 2 Руководство пользователя

После запуска программы можно изменить имена игроков. Первый игрок всегда играет за белых, второй за черных. Для этого нужно зайти в меню «Игра» и нажать кнопку «Изменить имена». (Рисунок 1).

После чего появится окно для смены имен. (Рисунок 2). Нажмите кнопку «ОК», когда закончите редактировать имена.

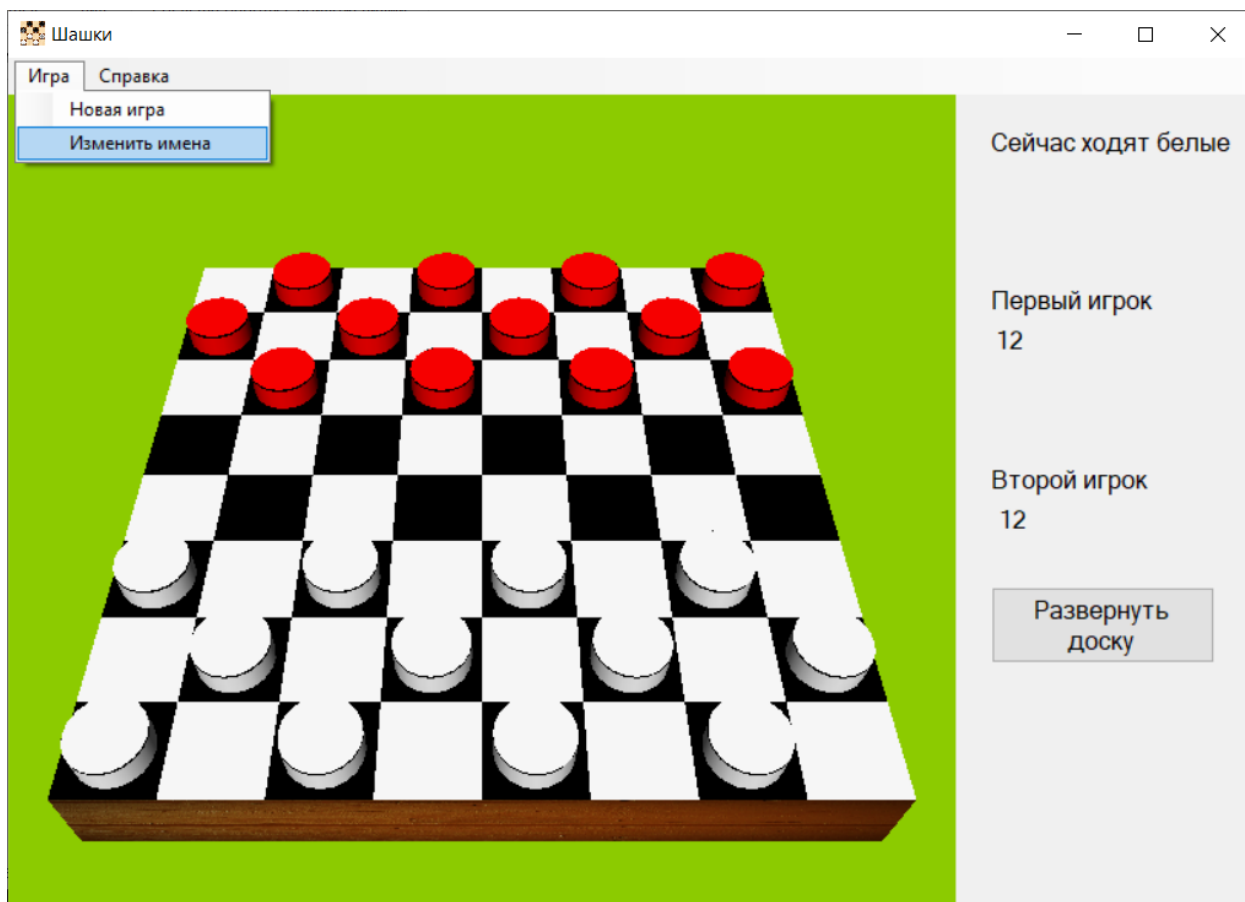


Рисунок 1 – Нажать кнопку «Изменить имена»

Поворачивать доску можно по трем осям. По оси Ох стрелками Вниз и Вверх. (Рисунок 3). По оси Оу стрелками влево, вправо. (Рисунок 4). По оси Oz латинскими буквами «А» и «D» (Рисунок 5).

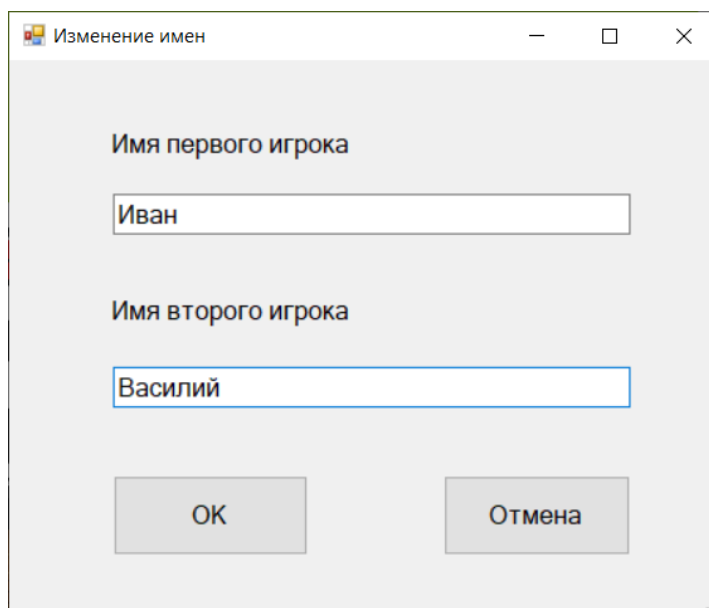


Рисунок 2 – Изменение имен

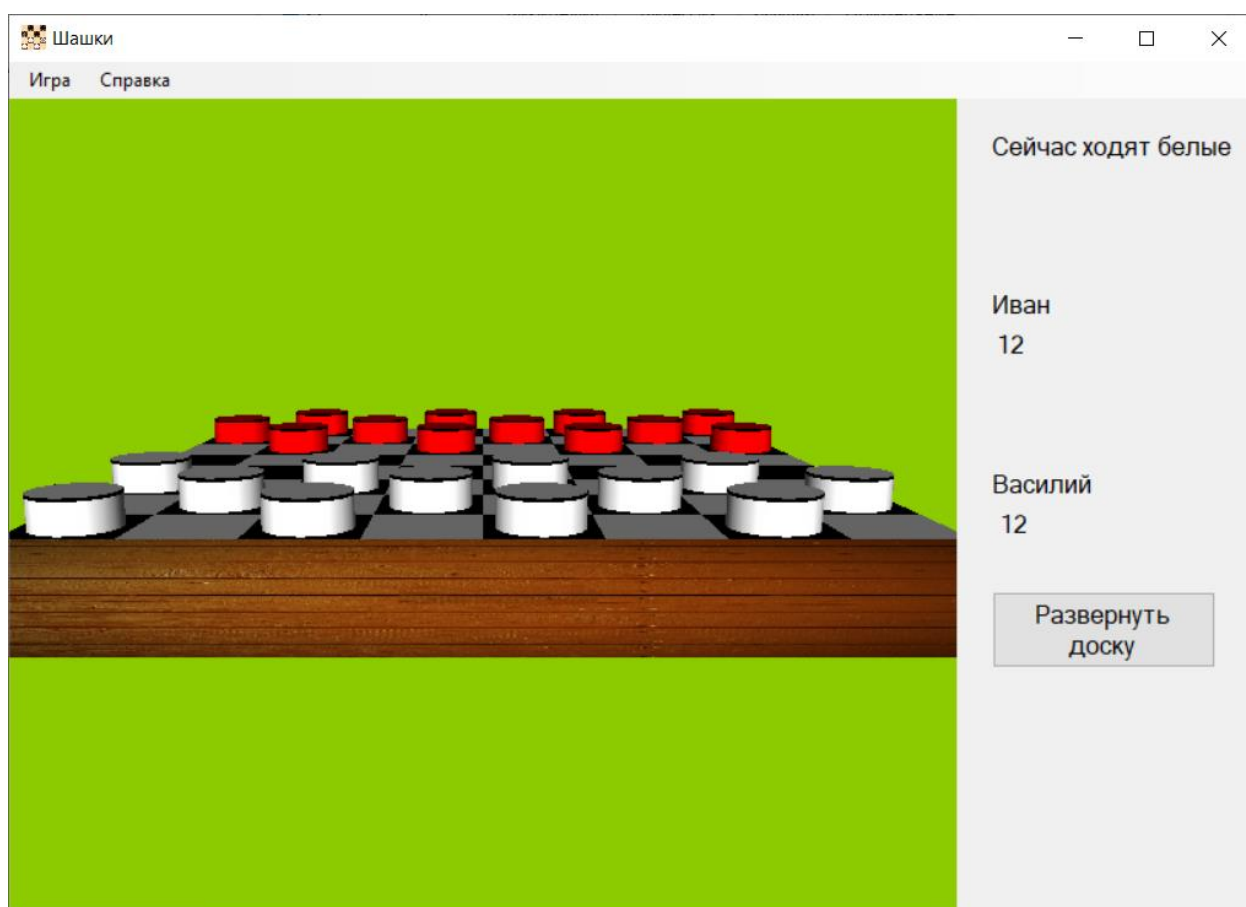


Рисунок 3 – Вращение по оси Ох



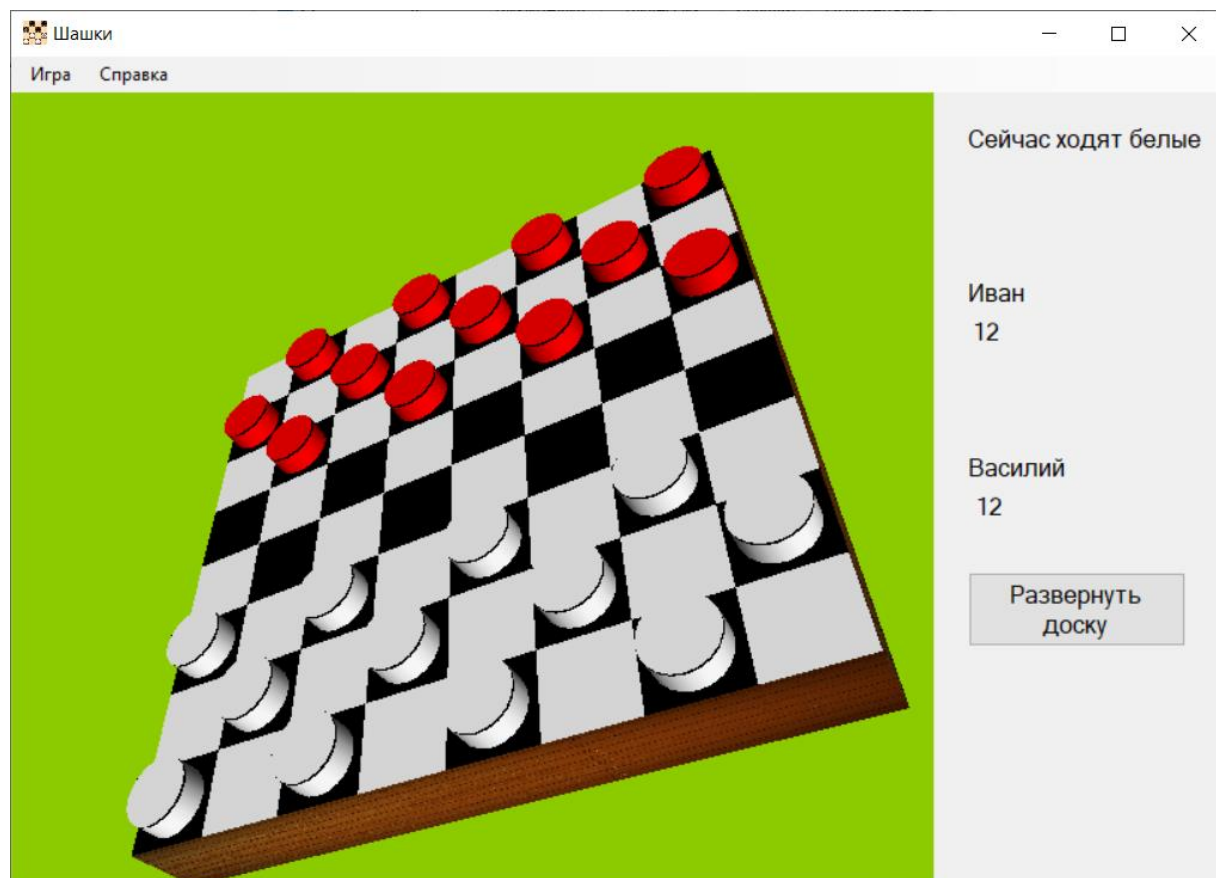


Рисунок 4 – Вращение по оси Oy

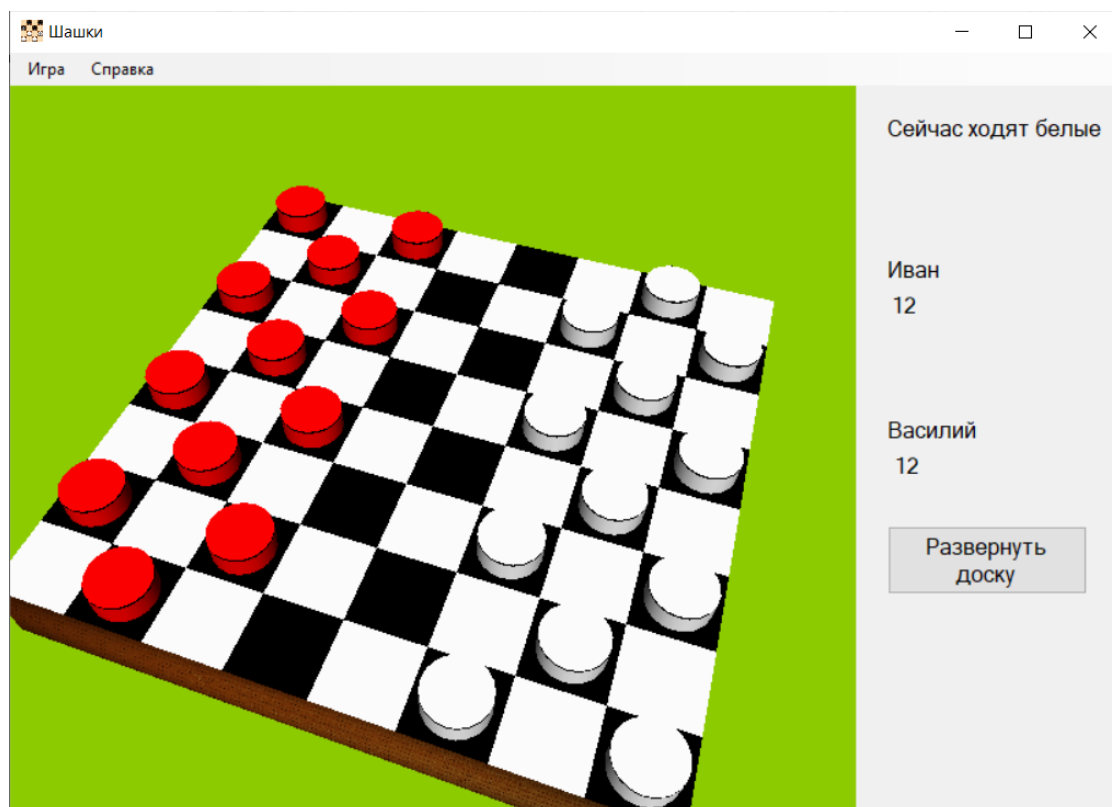


Рисунок 5 – Вращение по оси Oz

Также можно отдалять или приближать доску с шашками с помощью колесика мышки.

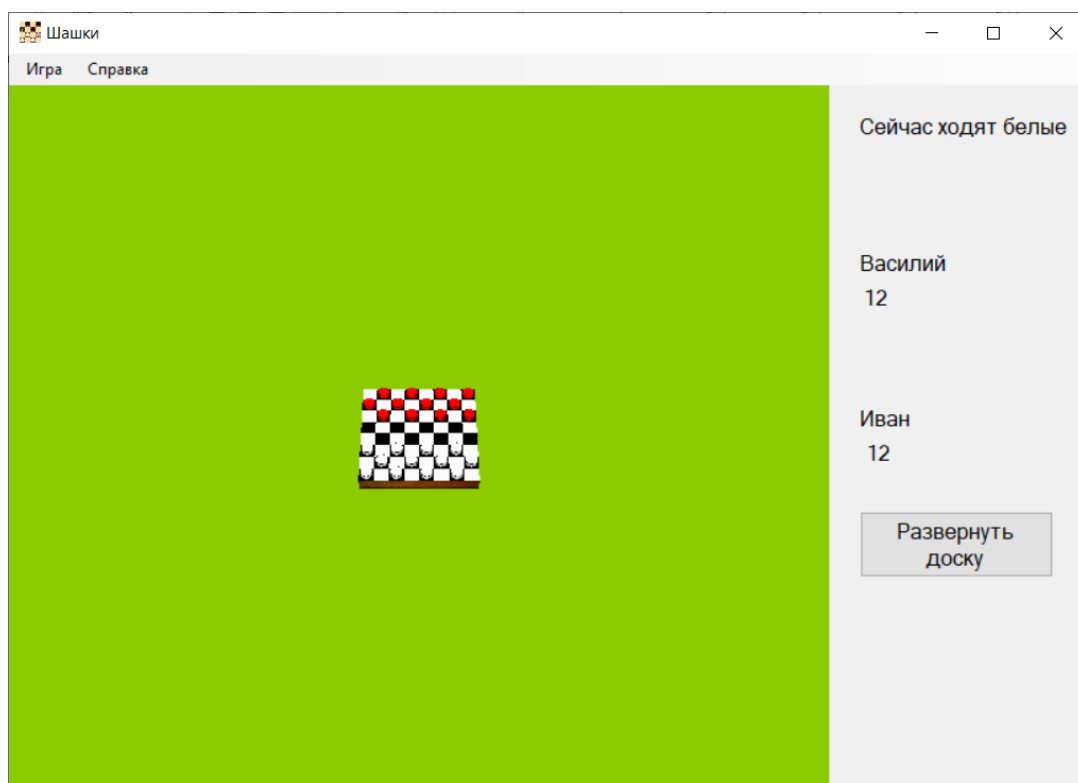


Рисунок 6 – Максимальное отдаление

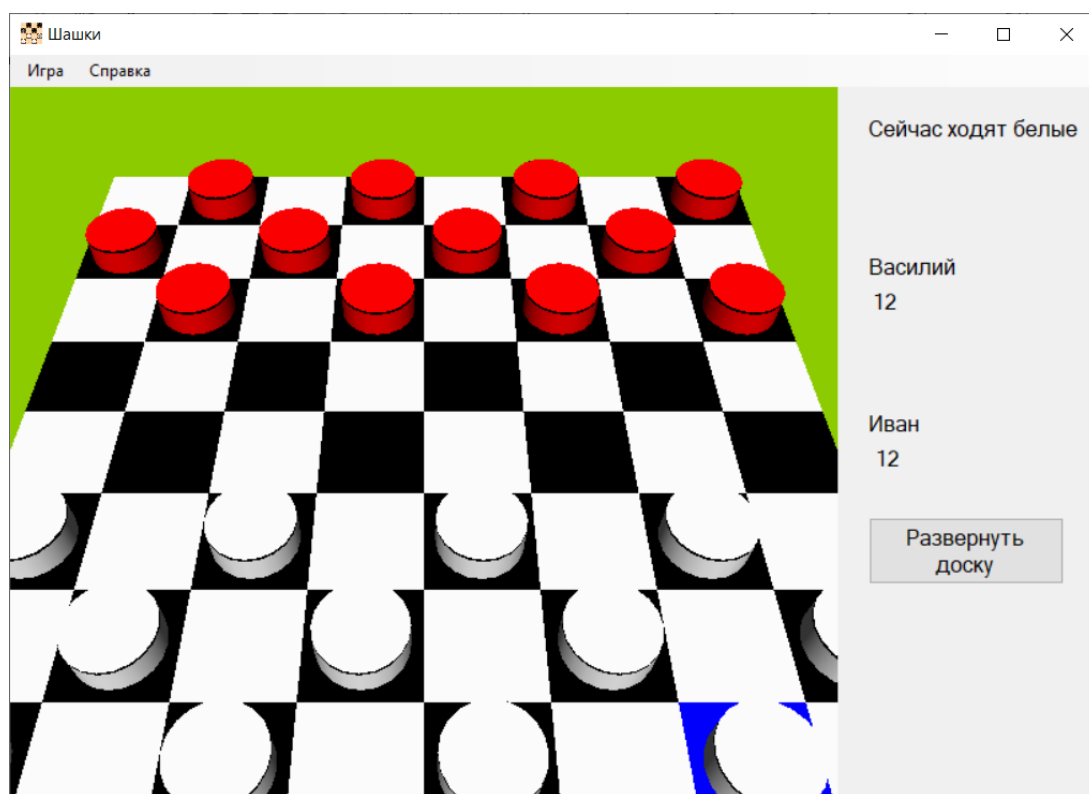


Рисунок 7 – Максимальное приближение

Синим цветом обозначается клетка, на которую наведен курсор. Для того чтобы выбрать шашку, достаточно щелкнуть по ней левой кнопкой. После чего желтым цветом будут показаны возможные ходы этой шашкой. (Рисунок 6). После чего можно выбрать, куда сходить шашкой, путем нажатия левой кнопки мыши по соответствующей желтой клетке. (Рисунок 7).

Затем право хода передается другому игроку. Отменить выбор шашки можно, щелкнув по ней еще раз. Если щелкнуть по другой своей шашке, выбор переключится на нее.

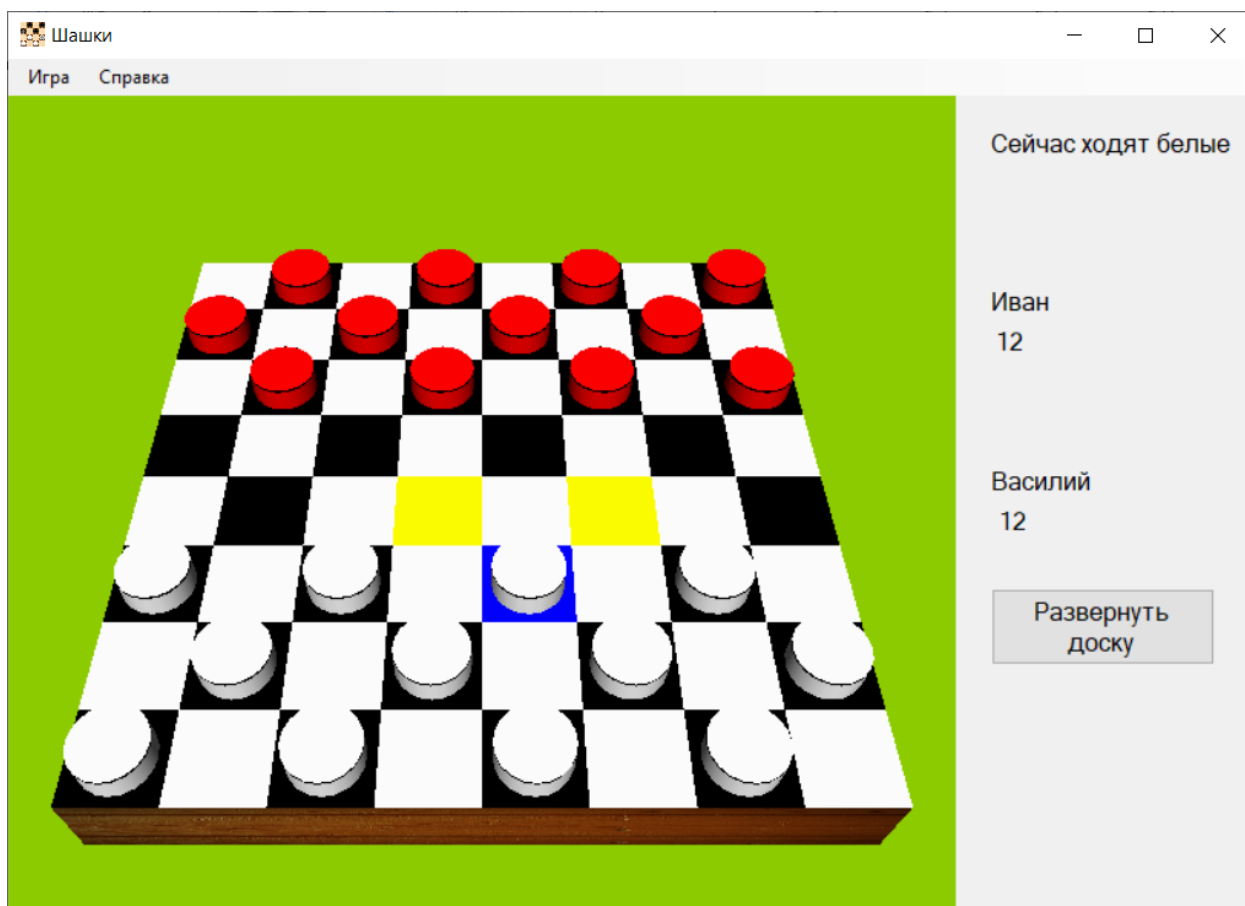


Рисунок 8 – Выбор шашки и хода

Если не получается выбрать определенную шашку, проверьте, ваш ли сейчас ход. Также может быть такая ситуация, когда какая-либо ваша шашка на данный момент может срубить шашку соперника. (Рисунок 10).

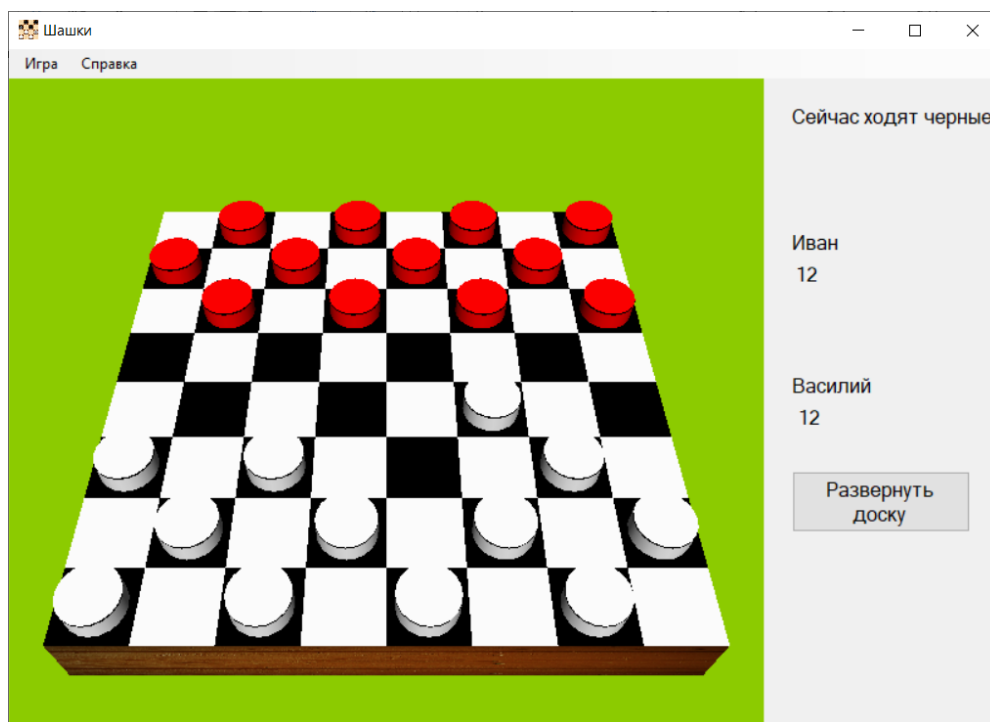


Рисунок 9 – Ход сделан

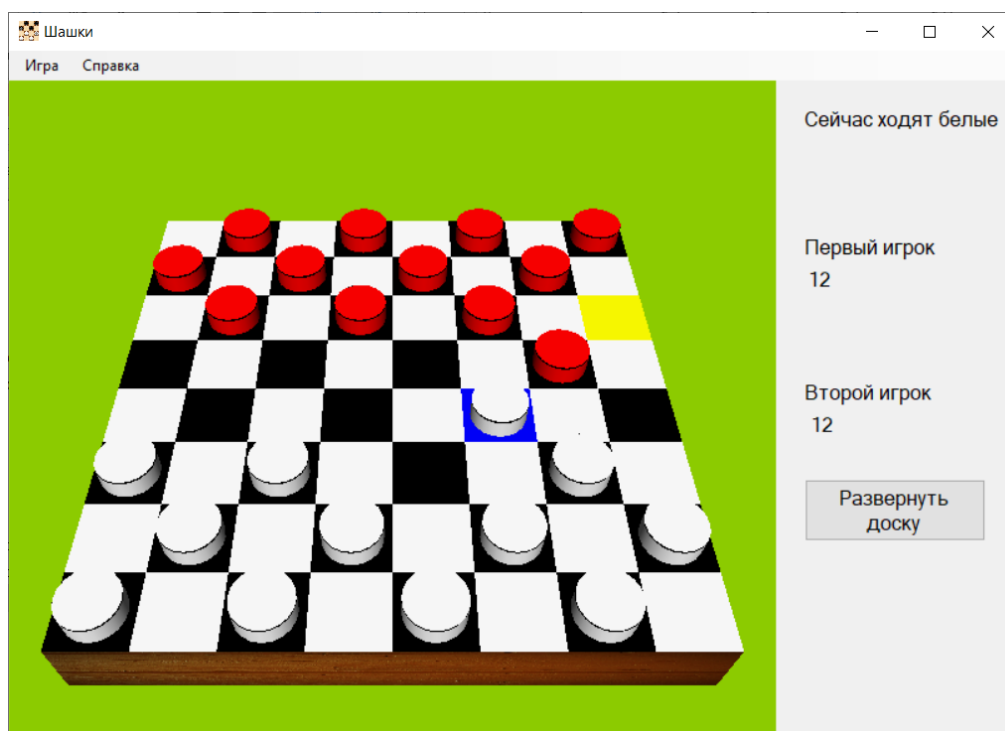


Рисунок 10 – Возможность рубить шашку

### ПРИЛОЖЕНИЕ 3 Результат выполнения программы

