**Министерство образования и науки РФ**

**ФГБОУ ВПО «ТАМБОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Системы автоматизированной поддержки принятия решений»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовой работе по дисциплине «ООП»

Студент Д.Ю. код 18 группа -21

фамилия, инициалы

1. Тема: Разработка игры «Шахматы» .

2. Срок представления проекта к защите « » .

3. Введение .

4. Основной раздел пояснительной записки .

5. Заключение .

6. Список используемой литературы .

7. Приложения .

Руководитель проекта (работы): В.И.

подпись, дата инициалы, фамилия

Задание принял к исполнению: Д.Ю.

подпись, дата инициалы, фамилия

Оценка: .

Дата: .

Тамбов

|  |  |  |
| --- | --- | --- |
|  | **СОДЕРЖАНИЕ** |  |
| 1. | Введение | 2 |
| 2. | Основной раздел | 3 |
| 2.1 | История игры | 3 |
| 2.2 | Объектная модель | 4 |
| 2.3 | Проектирование программного продукта | 5 |
| 2.4 | Тестирование программного продукта | 8 |
| 3. | Заключение | 9 |
| 4. | Список используемой литературы | 10 |
| 5. | Приложения | 11 |

1. **ВВЕДЕНИЕ**

Основой данной работы является создание игровой программы “шахматы” для игры между человеком и человеком. Так как реализовывать ИИ по заданию не требуется, то все сводится к следующему – создать доску, фигуры и реализовать алгоритм ходов всех типов фигур. Эти три задачи будут основой для дальнейшего построения макета приложения. В настоящее время написанием шахмат, если это не супер-пупер какие, можно и не увлекаться т.к. в свободном распространении есть игры на любой вкус и цвет. Работа преследует практические цели, а конкретно – разобраться с тонкостями используемого языка, применения парадигмы ооп как основы.

1. **ОСНОВНОЙ РАЗДЕЛ**
   1. **ИСТОРИЯ ИГРЫ**

Древнейшая форма шахмат — военная игра чатуранга — появилась в первые века н. э. В Индии чатурангой назывался вид войска, включавший боевые колесницы (ратха) – ладьи, слонов (хасти), конницу (ашва) и пеших воинов (падати). Игра символизировала битву с участием четырех родов войск, которыми управляет предводитель. Фигуры располагались по углам квадратной доски (аштапады) в 64 клетки, в игре принимало участие 4 человека. Движение фигур определялось бросанием костей. Чтобы выиграть в партии необходимо было уничтожить все войска противников. Чатуранга существовала в Индии до начала XX века, а название ее изменилось со временем на «чатурраджа» — игра четырех королей; фигуры же стали окрашиваться в 4 цвета — зеленый, желтый, красный и черный. Преемницей чатуранги стала игра шатранг (чатранг), возникшая в Центральной Азии в конце V — начале VI вв. В этой вариации игра имела два «лагеря» фигур и новую фигуру, изображающую советника короля — фарзина; в игре участие стали принимать уже лишь 2 соперника. Целью игры было поставить мат королю соперника. Таким образом «игру случая» заменила «игра разума». В VIII-IX вв. шатранг проник из Центральной Азии на Восток и Запад, ставший известным под арабским названием шатрандж. В шатрандже (IX-XV вв.) сохранилась терминология и расстановка фигур шатранга, однако внешний вид фигур претерпел изменения. Дело в том, что религия была против использования живых существ для обозначения шахматных фигур, поэтому арабы стали использовать в этих целях абстрактные фигурки в виде небольших цилиндров и конусов. Это значительно упростило их создание, что, в свою очередь, поспособствовало дальнейшему распространению игры в массах. Развитие игры проходило довольно медленно, так только ладья, король и конь ходили по современным правилам, диапазон же действия других фигур был крайне ограничен. Например, ферзь двигался только на одно поле по диагонали.

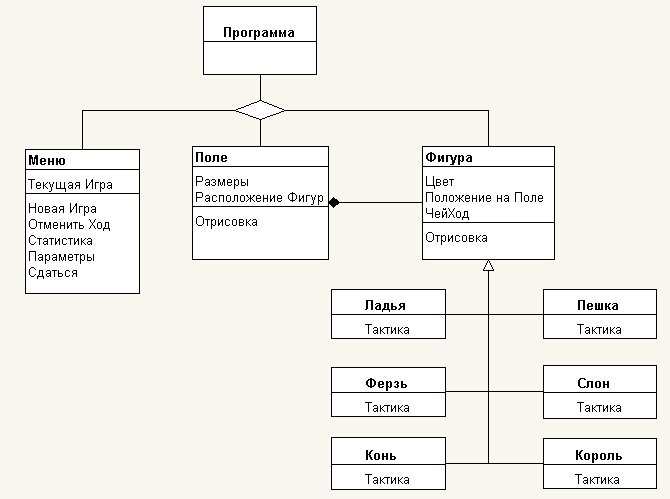
* 1. **ОБЪЕКТНАЯ МОДЕЛЬ**

Рисунок Объектная модель программы

На начальной стадии проектирования будущей программы определим главные составляющие – поле, фигура, меню (Рисунок 1.). Далее подгоняем эти три объекта к реальным шахматам. Получается, что поле должно содержать в себе некий набор свойств, ширину, длину, количество клеток, их цвет и расположение. Фигура как некое обобщение включает в себя только общее от шести, это цвет, расположение на доске, возможность хода (в зависимости от ситуаций в целом на поле). В тоже время фигура дробится на шесть более отличных друг от друга фигур, у каждой из шести есть своя тактика. Меню подразумевает набор стартовых точек, которые позволят более гибко управлять процессом игры.

Рисунок ОМТ

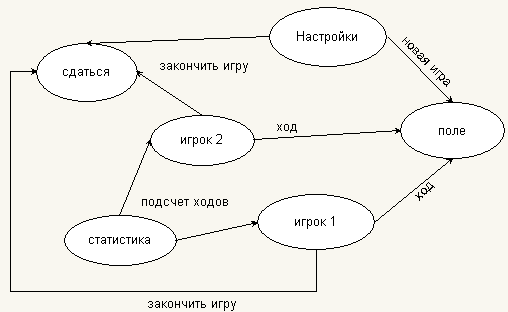
* 1. **ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА**

Рисунок Состояние

На этом этапе реализуем правила для игры, предполагаем что играют два игрока, очередность ходов складывается из того, кто играет за белую сторону (Рисунок 2). Каждый сделанный ход подразумевает изменение состояния объектов на поле, собирается некая статистика ходов за одну игру. Игра подразумевает под собой какое-то завершение, это может быть выигрыш одной из сторон (мат), сдача соперника (надоело играть), либо ничья. По достижении каждого из этих этапов ставится дальнейший выбор – выйти или начать заново.

В общем виде логика совершения хода фигурой следующая (Рисунок 3):

- берем объект Фигура и совершаем над ним действие (переход или взятие);

- результатом действия станет ошибка хода, либо смена положения объекта;

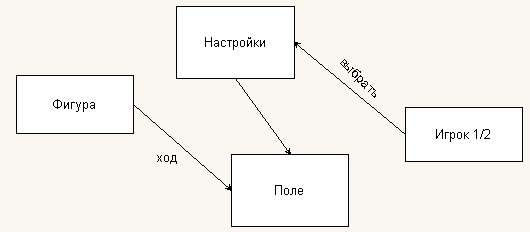


Рисунок Функциональная модель

**Рисунок 3 Логика хода фигурой**

Так как с классами определились, правила обрисовали, можно приступать к пробным играм. На рисунке 4 изображено подобие пробного хода в игре, здесь условная сторона делает ход одной из доступных фигур на поле и тут же показана возможность настройки составляющих игры, например выбор размера шахматного поля или сохранение состояния текущей игры.

Подсветка хода пешки следующая (Рисунок 5):

- проверить цвет фигуры;

- проверить поле два\_хода\_разрешены;

- сделать цветными панели +1 или +2 от положения пешки.

Рисунок Подсветка хода пешки

* 1. **ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА**

Игра начинается на чистом поле с расставленными на начальные позиции игровыми фигурами (Рисунок 6). По умолчанию первыми ходят черные, далее белые и так до завершения игры.

Рисунок Новая игра

Игровой процесс довольно просто настроить в разделах меню и окне настройки. Если по какой либо причине нет возможности управления мышью, то задаются клавиши. Если в момент игры вас прервали, то всегда можно сохранить свое текущее состояние в файл и закрыть приложение.

Правила игры такие же как если бы вы играли в обычные шахматы с другом. В том случае, если вы неуверенны в своих познаниях игровых правил, вам всегда подскажет правильное действие игровой помощник. В этом легко убедится совершив неправильный ход фигурой. Комфорт визуального восприятия процесса игры обеспечивает настройка отображения подложки доски и несколько пакетов с изображениями фигурок (Рисунок 7).

Рисунок Ход белых

Победа в игре достигается в том случае, когда главная по своему статусу фигура король одной из сторон попадает в ловушку, то есть невозможность сделать безопасный ход без угрозы быть взятым со стороны партнера.

1. **ЗАКЛЮЧЕНИЕ**

При написании данной работы, в первую очередь кода, я убедился в необходимости объектного моделирования поставленной задачи, ведь чем более масштабна цель, тем сложней ее выполнить «с нахрапа», а разбивая на подзадачи можно достичь быстроты, гибкости и простоты в реализации. Приложение «шахматы» показали, что грамотно составленная модель еще на первых стадиях проектировки в последующем позволит качественно воплощать даже непродуманные варианты игровых моментов.

1. **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**
2. C# 4.0 Полное руководство, авт. Г. Шилдт;
3. Библия C#, авт. М. Фленов;
4. Привила игры в шахматы, http://2ls.ru/pravila-shahmat/453-pravila-igry-v-shahmaty-dlya-novichkov.html
5. Парадигмы программирования, http://acm.mipt.ru/twiki/bin/view/Method/StylesSamples
6. **ПРИЛОЖЕНИЯ**
   1. **Класс fMain – стартовый в программе, визуальный процесс:**

Рисунок Классы

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Resources;

using System.Windows.Forms;

using DevComponents.DotNetBar;

using heaparessential.Controls;

using Шахматы.Properties;

/\*

FMain

\*menuStrip1

\*Panel1 \*tableLayoutPanel2

\*\*tableLayoutPanel1 \*\*lbl\_n

\*\*\*Bevel\_n

\*\*\*\*btn\_n

\*tableLayoutPanel3

\*\*lbl\_n

\*/

namespace Шахматы

{

public partial class fMain : Form

{

public fMain()

{

InitializeComponent();

NauStateKey();

//начальное положение квадрата

\_bevelStateIndex = "44";

//Подгрузим статистику из файла

\_saveToFile.Load();

\_formMenu.Statistic(\_saveToFile.Schetchik);

}

/// <summary>

/// Дефолтовые значения с XML

/// </summary>

private void NauStateKey()

{

\_keysB.Clear(); \_keysW.Clear();

#region key deffolt load from file xml

for (int i = 0; i <= 3; i++)

\_keysB.Add(\_кнопкиУправления.ReadParam("black", i).Key, \_кнопкиУправления.ReadParam("black", i).Value);

for (int i = 0; i <= 3; i++)

\_keysW.Add(\_кнопкиУправления.ReadParam("white", i).Key, \_кнопкиУправления.ReadParam("white", i).Value);

#endregion

}

private SortedList<string, string > \_keysB = new SortedList<string, string>();

private SortedList<string, string> \_keysW = new SortedList<string, string>();

/// <summary>

/// Коллекция обьектов Bevel

/// </summary>

private List<Bevel> \_bevels = new List<Bevel>();

/// <summary>

/// Коллекция обьектов Button

/// </summary>

private List<ButtonBase> \_buttons = new List<ButtonBase>();

private clNewGame newGame = new clNewGame();

private clWorkFile \_saveToFile = new clWorkFile();

private clStack \_stackXodov = new clStack();

private clCopyAsValue \_copyValueObj = new clCopyAsValue();

private fTransformPawnToN \_fTransformPawn = new fTransformPawnToN();

private void fMain\_Load(object sender, EventArgs e)

{

\_saveToFile.Schetchik += 1;

newGame = new clNewGame();

\_bevels.Clear();

for (int i = 0; i < tableLayoutPanel1.RowCount; i++)

for (int j = 0; j < tableLayoutPanel1.ColumnCount; j++)

if ((tableLayoutPanel1.GetControlFromPosition(i, j) as Bevel) != null)

\_bevels.Add((tableLayoutPanel1.GetControlFromPosition(i, j) as Bevel));

\_buttons.Clear();

foreach (Bevel bevel in \_bevels.Where(bevel => (bevel.GetChildAtPoint(new Point(0, 0)) as ButtonBase) != null))

{

\_buttons.Add(bevel.GetChildAtPoint(new Point(0, 0)) as ButtonBase);

\_copyValueObj.AddList(bevel.GetChildAtPoint(new Point(0, 0)) as ButtonBase);

}

}

/\*КТО ИДЕТ за\_клинским ПЕРВЫМ?\*/

private clXod xod = new clXod(true);

/// <summary>

/// Варианты действий обьекта

/// </summary>

private void xod\_Click(object sender, EventArgs e)

{

#region Button\_Click

//убедимся что нажали кнопку

if ((sender as ButtonBase) != null)

{

var btnFigure = (sender as ButtonBase);//текущая фигура

//ЩА ХОДЯТ ЧЕРНЫЕ:

if (xod.BlackGo && btnFigure.Tag.ToString().Contains("ЧЕР"))

{//нажали не туже самую кнопку повторно

if (xod.XodStartFigureTag != btnFigure.Tag.ToString())

{

/\*Методы отвечающие за начало хода фигурой\*/

/\*Нажал кнопу, снял подсветку, подсветил ходы\*/

//MessageBox.Show("Начало хода ЭТОЙ кнопкой ИЛИ буду бить ЭТУ кнопу");

\_btnBase = btnFigure;

noCheked();

podsvetXoda(btnFigure.Tag.ToString(), btnFigure.Name);

/\*Методы отвечающие за возможность битья

\* если текущая кнопка под ударом предыдущей

\*/

xod.XodStart = true;

xod.XodStartFigureTag = btnFigure.Tag.ToString();

}

}

else if (xod.BlackGo && btnFigure.Tag.ToString().Contains("БЕЛ"))

{

if (string.IsNullOrEmpty(xod.XodStartFigureTag))

{

showBall(xod.BlackGo ? "errBlackGo" : "errWhiteGo");

return;

}

//заряжаю тег кнопки "ЧЕР\_СЛОН\_A2", получаю индекс [6,0] в массиве

Point indexStart = newGame.PoiskVMassive(btnFigure.Tag.ToString()); //[6,0]

string bvl = indexStart.X + "" + indexStart.Y;

string result = newGame.VariantXoda(xod.XodStartFigureTag, bvl, true);

//Могу ее(бел) выбить?

switch (result)

{

case "help": showBall(xod.ConteinsFigure(xod.XodStartFigureTag));

break;

case "Путь перекрыт": showBall("errNe");

break;

case "err": showBall(xod.BlackGo ? "errBlackGo" : "errWhiteGo");

break;

case "OK":// MessageBox.Show("БИТЬ МОЖНО");

deliteFigure(btnFigure, xod.XodStartFigureTag);

xod.BlackGo = !xod.BlackGo;

xod.XodStart = false;

xod.XodStartFigureTag = "";

//\_stackXodov.Add(newGame.FigureObjArrState(), xod.BlackGo);

\_stackXodov.AddString(newGame.FigureObjArrState(), xod.BlackGo);

return;

}

}

//ЩА ХОДЯТ БЕЛЫЕ:

if (!xod.BlackGo && btnFigure.Tag.ToString().Contains("БЕЛ"))

{//нажали не туже самую кнопку повторно

if (xod.XodStartFigureTag != btnFigure.Tag.ToString())

{

/\*Методы отвечающие за начало хода фигурой\*/

/\*Нажал кнопу, снял подсветку, подсветил ходы\*/

//MessageBox.Show("Начало хода ЭТОЙ кнопкой ИЛИ буду бить ЭТУ кнопу");

\_btnBase = btnFigure;

noCheked();

podsvetXoda(btnFigure.Tag.ToString(), btnFigure.Name);

/\*Методы отвечающие за возможность битья

\* если текущая кнопка под ударом предыдущей

\*/

xod.XodStart = true;

xod.XodStartFigureTag = btnFigure.Tag.ToString();

}

}

else if (!xod.BlackGo && btnFigure.Tag.ToString().Contains("ЧЕР"))

{

if (string.IsNullOrEmpty(xod.XodStartFigureTag))

{

showBall(xod.BlackGo ? "errBlackGo" : "errWhiteGo");

return;

}

//заряжаю тег кнопки "ЧЕР\_СЛОН\_A2", получаю индекс [6,0] в массиве

Point indexStart = newGame.PoiskVMassive(btnFigure.Tag.ToString()); //[6,0]

string bvl = indexStart.X + "" + indexStart.Y;

string result = newGame.VariantXoda(xod.XodStartFigureTag, bvl, true);

//Могу ее(чер) выбить?

switch (result)

{

case "help": showBall(xod.ConteinsFigure(xod.XodStartFigureTag));

break;

case "Путь перекрыт": showBall("errNe");

break;

case "err": showBall(xod.BlackGo ? "errBlackGo" : "errWhiteGo");

break;

case "OK":// MessageBox.Show("БИТЬ МОЖНО");

deliteFigure(btnFigure, xod.XodStartFigureTag);

ProverkaNaShax(xod.XodStartFigureTag);

xod.BlackGo = !xod.BlackGo;

xod.XodStart = false;

xod.XodStartFigureTag = "";

//\_stackXodov.Add(newGame.FigureObjArrState(), xod.BlackGo);

\_stackXodov.AddString(newGame.FigureObjArrState(), xod.BlackGo);

return;

}

}

}

#endregion

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#region Bevel

if ((sender as Bevel) != null)

{

//Если начала хода небыло

if(xod.XodStart)

{

/\*Методы отвечающие за переход кнопки В панель

\* на форме

\* в массиве

\*/

var bvl = (sender as Bevel);

//если кнопке xod.XodStartFigureTag разрешен переход на панель bvl.Tag:

//если на панели есть фигура, то бить ИЛИ (это своя, туда нельзя)

string result = newGame.VariantXoda(xod.XodStartFigureTag, bvl.Tag.ToString(), false);

switch (result)

{

case "help": showBall(xod.ConteinsFigure(xod.XodStartFigureTag));

break;

case "Путь перекрыт": showBall("errNe");

break;

default:

{

perexod(bvl);

ProverkaNaShax(xod.XodStartFigureTag);

xod.BlackGo = !xod.BlackGo;

//\_stackXodov.Add(newGame.FigureObjArrState(), xod.BlackGo);

\_stackXodov.AddString(newGame.FigureObjArrState(), xod.BlackGo);

}break;

}

xod.XodEnd = true;

}

}

#endregion

//Ход совершен полностью

xod.ResetXodState();

}

/// <summary>

/// Показать варианты хода для фигуры btnFigureTag

/// </summary>

private void podsvetXoda(string btnFigureTag, string name)

{

if(\_formMenu.PodsvetOn)

switch (name)

{

#region case "btnBP\_a2": GoPawn(btnFigureTag); break;

//если тег кнопки изменен (ПЕШКА \_ КОНЬ) то это КОНЬ

case "btnBP\_a2": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnBP\_b2": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnBP\_c2": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnBP\_d2": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnBP\_e2": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnBP\_f2": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnBP\_g2": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnBP\_h2": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

#endregion

#region case "btnWP\_a7": GoPawn(btnFigureTag); break;

case "btnWP\_a7": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnWP\_b7": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnWP\_c7": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnWP\_d7": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnWP\_e7": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnWP\_f7": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnWP\_g7": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

case "btnWP\_h7": if (btnFigureTag.Contains("ПЕШКА")) GoPawn(btnFigureTag);

else podsvetXoda(btnFigureTag, TagName(btnFigureTag));

break;

#endregion

#region case "btnBL\_a1": GoLaden(btnFigureTag); break;

case "btnBL\_a1":

GoLaden(btnFigureTag); break;

case "btnBL\_h1":

GoLaden(btnFigureTag); break;

#endregion

#region case "btnWL\_a8": GoLaden(btnFigureTag); break;

case "btnWL\_a8":

GoLaden(btnFigureTag); break;

case "btnWL\_h8":

GoLaden(btnFigureTag); break;

#endregion

#region case "btnBK\_b1": GoKoni(btnFigureTag); break;

case "btnBK\_b1": GoKoni(btnFigureTag); break;

case "btnBK\_g1": GoKoni(btnFigureTag); break;

#endregion

#region case "btnWK\_b8": GoKoni(btnFigureTag); break;

case "btnWK\_b8": GoKoni(btnFigureTag); break;

case "btnWK\_g8": GoKoni(btnFigureTag); break;

#endregion

#region case "btnBS\_c1": GoSlon(btnFigureTag); break;

case "btnBS\_c1": GoSlon(btnFigureTag); break;

case "btnBS\_f1": GoSlon(btnFigureTag); break;

#endregion

#region case "btnWS\_c8": GoSlon(btnFigureTag); break;

case "btnWS\_c8": GoSlon(btnFigureTag); break;

case "btnWS\_f8": GoSlon(btnFigureTag); break;

#endregion

case "btnBW\_d1": GoKarol(btnFigureTag); break;

case "btnWW\_d8": GoKarol(btnFigureTag); break;

case "btnBF\_e1": GoFerz(btnFigureTag); break;

case "btnWF\_e8": GoFerz(btnFigureTag); break;

case "err": throw new Exception("Ошибка TagName() в преобразовании фигуры");

}

}

/// <summary>

/// выдать направление на swith выше.

/// на вход "ЧЕР\_btnIKoni\_1" на выход "btnBK\_b1"

/// </summary>

private string TagName(string name)

{

string[] tmp = name.Split('\_');

if(name.Contains("ЧЕР"))

{

switch (tmp[1])

{

case "btnILaden": return "btnBL\_a1";

case "btnIKoni": return "btnBK\_b1";

case "btnISlon": return "btnBS\_c1";

case "btnIFerz": return "btnBF\_e1";

}

}

else

{

switch (tmp[1])

{

case "btnILaden": return "btnWL\_a8";

case "btnIKoni": return "btnWK\_b8";

case "btnISlon": return "btnWS\_c8";

case "btnIFerz": return "btnWF\_e8";

}

}

return "err";

}

/// <summary>

/// Для ферзя подсветка хода

/// </summary>

private void GoFerz(string btnFigureTag)

{

/\*

\*Ферзь - это направления:

\*а) Слон

\*б) Ладья

\*используем

\*/

GoLaden(btnFigureTag);

GoSlon(btnFigureTag);

}

/// <summary>

/// Для короля подсветка хода

/// </summary>

private void GoKarol(string btnFigureTag)

{

/\*варианты обхода:

\*3-3-1-1

\*2-2-2-2

\*верх-низ-лево-право

\*/

//чем ходим

bool goBlack = btnFigureTag.Contains("ЧЕР");

bool goWhite = btnFigureTag.Contains("БЕЛ");

//заряжаю тег кнопки "ЧЕР\_КАРОЛЬ\_A2", получаю индекс [6,0] в массиве

Point indexStart = newGame.PoiskVMassive(btnFigureTag);//[6,0]

const int up = 0, down = 7, left = 0, right = 7;

string indexL = "", indexR = "", indexC = "";

#region Верхнее направление

//поднялись по строке

int goUp = indexStart.X - 1;

//по столбцам

int goLeft = indexStart.Y - 1;

int goCentr = indexStart.Y;

int goRight = indexStart.Y + 1;

if (goLeft >= left) indexL = goUp + "" + goLeft;

if (goRight <= right) indexR = goUp + "" + goRight;

if (goCentr >= up) indexC = goUp + "" + goCentr;

//убедится что не нарушены границы доски

if (goUp >= up && (indexL != "" || indexR != ""))

{

string figNameL = newGame.ShowFigName(indexL);

string figNameC = newGame.ShowFigName(indexC);

string figNameR = newGame.ShowFigName(indexR);

//разрешено вверх и (вправо или влево)

foreach (var bevel in \_bevels)

{

//"40" = "40"

if (bevel.Tag.ToString() == indexL)//|| bevel.Tag.ToString() == indexR)

{

if (goBlack && (figNameL == "null" || figNameL.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameL == "null" || figNameL.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

if (bevel.Tag.ToString() == indexC)

{

if (goBlack && (figNameC == "null" || figNameC.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameC == "null" || figNameC.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

if (bevel.Tag.ToString() == indexR)

{

if (goBlack && (figNameR == "null" || figNameR.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameR == "null" || figNameR.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

}

}

indexL = indexR = "";

#endregion

#region Левое направление

goLeft = indexStart.Y - 1;

if(goLeft>=left)

{

indexL = indexStart.X + "" + goLeft;

string figNameL = newGame.ShowFigName(indexL);

foreach(Bevel bvl in \_bevels)

if (bvl.Tag.ToString() == indexL)//|| bevel.Tag.ToString() == indexR)

{

if (goBlack && (figNameL == "null" || figNameL.Contains("БЕЛ")))

bvl.BackColor = Color.Green;

if (goWhite && (figNameL == "null" || figNameL.Contains("ЧЕР")))

bvl.BackColor = Color.Green;

}

}

#endregion

#region Правое направление

goRight = indexStart.Y + 1;

if (goRight <= right)

{

indexR = indexStart.X + "" + goRight;

string figNameR = newGame.ShowFigName(indexR);

foreach (Bevel bvl in \_bevels)

if (bvl.Tag.ToString() == indexR)

{

if (goBlack && (figNameR == "null" || figNameR.Contains("БЕЛ")))

bvl.BackColor = Color.Green;

if (goWhite && (figNameR == "null" || figNameR.Contains("ЧЕР")))

bvl.BackColor = Color.Green;

}

}

#endregion

#region Нижнее направление

//опустились по строке

goUp = indexStart.X + 1;

//по столбцам

goLeft = indexStart.Y - 1;

goCentr = indexStart.Y;

goRight = indexStart.Y + 1;

if (goLeft >= left) indexL = goUp + "" + goLeft;

if (goRight <= right) indexR = goUp + "" + goRight;

if (goCentr <= down) indexC = goUp + "" + goCentr;

//убедится что не нарушены границы доски

if (goUp <= down && (indexL != "" || indexR != ""))

{//goUp типо goDown ну лень

string figNameL = newGame.ShowFigName(indexL);

string figNameC = newGame.ShowFigName(indexC);

string figNameR = newGame.ShowFigName(indexR);

//разрешено вверх и (вправо или влево)

foreach (var bevel in \_bevels)

{

//"40" = "40"

if (bevel.Tag.ToString() == indexL)//|| bevel.Tag.ToString() == indexR)

{

if (goBlack && (figNameL == "null" || figNameL.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameL == "null" || figNameL.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

if (bevel.Tag.ToString() == indexC)

{

if (goBlack && (figNameC == "null" || figNameC.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameC == "null" || figNameC.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

if (bevel.Tag.ToString() == indexR)

{

if (goBlack && (figNameR == "null" || figNameR.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameR == "null" || figNameR.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

}

}

#endregion

}

/// <summary>

/// Для слона подсветка хода

/// </summary>

private void GoSlon(string btnFigureTag)

{

//чем ходим

bool goBlack = btnFigureTag.Contains("ЧЕР");

bool goWhite = btnFigureTag.Contains("БЕЛ");

//заряжаю тег кнопки "ЧЕР\_СЛОН\_A2", получаю индекс [6,0] в массиве

Point indexStart = newGame.PoiskVMassive(btnFigureTag); //[6,0]

const int up = 0, down = 7, left = 0, right = 7;

/\*направления хода: границы//верх=0//низ=7//лево=0//право=7

\*

\* в процессе хода незаходить за обьект

\*/

string indexL = "", indexR = "";

#region Верхнее направление

bool rstop = false, lstop = false;

//глянем в право

int goRight = indexStart.Y + 1;

//глянем в лево

int goLeft = indexStart.Y - 1;

//поднятся на верх

int goUp = indexStart.X - 1;

//проверить на выход за грань

while (goUp >= up)

{

//мы ненарушили и поднялись

//проверить на выход за грань

if (goLeft >= left)

{

//мы ненарушили и перешли в лево, координаты:

indexL = goUp + "" + goLeft;

//глянем наличие обьекта по этим координатам в массиве:

string figNameL = newGame.ShowFigName(indexL);

foreach (var bevel in \_bevels)

{

if (!lstop && bevel.Tag.ToString() == indexL)

{

if (goBlack && figNameL.Contains("ЧЕР"))

lstop = true;

if (goWhite && figNameL.Contains("БЕЛ"))

lstop = true;

if (goBlack && figNameL.Contains("БЕЛ"))

{ bevel.BackColor = Color.Green; lstop = true; }

if (goWhite && figNameL.Contains("ЧЕР"))

{ bevel.BackColor = Color.Green; lstop = true; }

if (goBlack && (figNameL == "null" || figNameL.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameL == "null" || figNameL.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

}

goLeft--;

}

//проверить на выход за грань

if (goRight <= right)

{

//мы ненарушили и перешли в право, координаты:

indexR = goUp + "" + goRight;

//глянем наличие обьекта по этим координатам в массиве:

string figNameR = newGame.ShowFigName(indexR);

foreach (var bevel in \_bevels)

{

if (!rstop && bevel.Tag.ToString() == indexR)

{

if (goBlack && figNameR.Contains("ЧЕР"))

rstop = true;

if (goWhite && figNameR.Contains("БЕЛ"))

rstop = true;

if (goBlack && figNameR.Contains("БЕЛ"))

{ bevel.BackColor = Color.Green; rstop = true; }

if (goWhite && figNameR.Contains("ЧЕР"))

{ bevel.BackColor = Color.Green; rstop = true; }

if (goBlack && (figNameR == "null" || figNameR.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameR == "null" || figNameR.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

}

goRight++;

}

goUp--;

}

indexL = indexR = "";

#endregion

#region Нижнее направление

rstop = lstop = false;

//глянем в право

goRight = indexStart.Y + 1;

//глянем в лево

goLeft = indexStart.Y - 1;

//опустится в низ

goUp = indexStart.X + 1;

//проверить на выход за грань

while (goUp <= down)

{//goUp насамом деле goDown, лень...

//мы ненарушили и опустились

//проверить на выход за грань

if (goLeft >= left)

{

//мы ненарушили и перешли в лево, координаты:

indexL = goUp + "" + goLeft;

//глянем наличие обьекта по этим координатам в массиве:

string figNameL = newGame.ShowFigName(indexL);

foreach (var bevel in \_bevels)

{

if (!lstop && bevel.Tag.ToString() == indexL)

{

if (goBlack && figNameL.Contains("ЧЕР"))

lstop = true;

if (goWhite && figNameL.Contains("БЕЛ"))

lstop = true;

if (goBlack && figNameL.Contains("БЕЛ"))

{ bevel.BackColor = Color.Green; lstop = true; }

if (goWhite && figNameL.Contains("ЧЕР"))

{ bevel.BackColor = Color.Green; lstop = true; }

if (goBlack && (figNameL == "null" || figNameL.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameL == "null" || figNameL.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

}

goLeft--;

}

//проверить на выход за грань

if (goRight <= right)

{

//мы ненарушили и перешли в право, координаты:

indexR = goUp + "" + goRight;

//глянем наличие обьекта по этим координатам в массиве:

string figNameR = newGame.ShowFigName(indexR);

foreach (var bevel in \_bevels)

{

if (!rstop && bevel.Tag.ToString() == indexR)

{

if (goBlack && figNameR.Contains("ЧЕР"))

rstop = true;

if (goWhite && figNameR.Contains("БЕЛ"))

rstop = true;

if (goBlack && figNameR.Contains("БЕЛ"))

{ bevel.BackColor = Color.Green; rstop = true; }

if (goWhite && figNameR.Contains("ЧЕР"))

{ bevel.BackColor = Color.Green; rstop = true; }

if (goBlack && (figNameR == "null" || figNameR.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameR == "null" || figNameR.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

}

goRight++;

}

goUp++;

}

indexL = indexR = "";

#endregion

}

/// <summary>

/// Для коня подсветка хода

/// </summary>

private void GoKoni(string btnFigureTag)

{

//чем ходим

bool goBlack = btnFigureTag.Contains("ЧЕР");

bool goWhite = btnFigureTag.Contains("БЕЛ");

//заряжаю тег кнопки "ЧЕР\_КОНЬ\_A2", получаю индекс [6,0] в массиве

Point indexStart = newGame.PoiskVMassive(btnFigureTag);//[6,0]

const int up = 0, down = 7, left = 0, right = 7;

/\*направления хода: границы//верх=0//низ=7//лево=0//право=7

\*

\* в процессе хода незаходить за обьект

\*/

string indexL="", indexR="";

#region Верхнее направление

//по строке

int goUp = indexStart.X - 2;//6-2

//по столбцам

int goLeft = indexStart.Y - 1;

int goRight = indexStart.Y + 1;

if(goLeft>=left) indexL = goUp + "" + goLeft;

if(goRight<=right) indexR = goUp + "" + goRight;

//убедится что не нарушены границы доски

if (goUp >= up && (indexL != "" || indexR != ""))

{

string figNameL = newGame.ShowFigName(indexL);

string figNameR = newGame.ShowFigName(indexR);

//разрешено вверх и (вправо или влево)

foreach (var bevel in \_bevels)

{

//"40" = "40"

if (bevel.Tag.ToString() == indexL)//|| bevel.Tag.ToString() == indexR)

{

if (goBlack && (figNameL == "null" || figNameL.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameL == "null" || figNameL.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

if (bevel.Tag.ToString() == indexR)

{

if (goBlack && (figNameR == "null" || figNameR.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameR == "null" || figNameR.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

}

}

indexL = indexR = "";

#endregion

#region Нижнее направление

//по строке

goUp = indexStart.X + 2;//6+2

//по столбцам

goLeft = indexStart.Y - 1;

goRight = indexStart.Y + 1;

if (goLeft >= left) indexL = goUp + "" + goLeft;

if (goRight <= right) indexR = goUp + "" + goRight;

//убедится что не нарушены границы доски

if (goUp <= down && (indexL != "" || indexR != ""))

{

string figNameL = newGame.ShowFigName(indexL);

string figNameR = newGame.ShowFigName(indexR);

//разрешено вниз и (вправо или влево)

foreach (var bevel in \_bevels)

{

//"40" = "40"

if (bevel.Tag.ToString() == indexL)

{

if (goBlack && (figNameL == "null" || figNameL.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameL == "null" || figNameL.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

if (bevel.Tag.ToString() == indexR)

{

if (goBlack && (figNameR == "null" || figNameR.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameR == "null" || figNameR.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

}

}

indexL = indexR = "";

#endregion

#region Влево направление

//по строке вверх и вниз

goUp = indexStart.X + 1;//до 7

int goDown = indexStart.X - 1;//до 0

//по столбцам в лево

goLeft = indexStart.Y - 2;//до 0

if(goUp<=down) indexL = goUp+""+goLeft;

if(goDown>=up) indexR = goDown+""+goLeft;

//убедится что не нарушены границы доски

if (goLeft>=left && (indexL != "" || indexR != ""))

{

string figNameL = newGame.ShowFigName(indexL);

string figNameR = newGame.ShowFigName(indexR);

//разрешено вдево и (выше или ниже)

foreach (var bevel in \_bevels)

{

//"40" = "40"

if (bevel.Tag.ToString() == indexL)//верх

{

if (goBlack && (figNameL == "null" || figNameL.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameL == "null" || figNameL.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

if (bevel.Tag.ToString() == indexR)//низ

{

if (goBlack && (figNameR == "null" || figNameR.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameR == "null" || figNameR.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

}

}

indexL = indexR = "";

#endregion

#region Вправо направление

//по строке вверх и вниз

goUp = indexStart.X + 1;//до 7

goDown = indexStart.X - 1;//до 0

//по столбцам в право

goRight = indexStart.Y + 2;//до 7

if(goUp<=down) indexL = goUp+""+goRight;

if(goDown>=up) indexR = goDown+""+goRight;

//убедится что не нарушены границы доски

if (goRight<=right && (indexL != "" || indexR != ""))

{

string figNameL = newGame.ShowFigName(indexL);

string figNameR = newGame.ShowFigName(indexR);

//разрешено вдево и (выше или ниже)

foreach (var bevel in \_bevels)

{

//"40" = "40"

if (bevel.Tag.ToString() == indexL)//|| bevel.Tag.ToString() == indexR)

{

if (goBlack && (figNameL == "null" || figNameL.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameL == "null" || figNameL.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

if (bevel.Tag.ToString() == indexR)

{

if (goBlack && (figNameR == "null" || figNameR.Contains("БЕЛ")))

bevel.BackColor = Color.Green;

if (goWhite && (figNameR == "null" || figNameR.Contains("ЧЕР")))

bevel.BackColor = Color.Green;

}

}

}

indexL = indexR = "";

#endregion

}

/// <summary>

/// Для ладьи подсветка хода

/// </summary>

private void GoLaden(string btnFigureTag)

{

bool goBlack = btnFigureTag.Contains("ЧЕР");

//заряжаю тег кнопки "ЧЕР\_ЛАДЬЯ\_A2", получаю индекс [6,0] в массиве

Point indexStart = newGame.PoiskVMassive(btnFigureTag);//[6,0]

//индекс [6,0] в строку "60", строку ищем в Bevel.Tag

//string nn = (indexStart.X) + "" + (indexStart.Y);//6+""+0

const int up = 0, down = 7, left = 0, right = 7;

/\*направления хода: границы//верх=0//низ=7//лево=0//право=7

\* в верх: от [6-1, 2] до [верх, 2]

\* в низ : от [6+1, 2] до [низ, 2]

\* в лево: от [6, 2-1] до [6, лево]

\*в право: от [6, 2+1] до [6, право]

\*

\* в процессе хода незаходить за обьект

\*/

List<string > toUp = new List<string>();

#region Верхнее направление

for (int i = (indexStart.X - 1); i >= up; i--)

{//6..5..4..0

string tmp = (i) + "" + indexStart.Y;

string figName = newGame.ShowFigName(tmp);

//если ячейка пуста, то подсветим:

if (figName == "null") toUp.Add(tmp);

if (goBlack && figName != "null")

{

//если в ячейке обьект И я пошел ЧЕРными И обьект\_БЕЛ, то подсветим

if (figName.Contains("БЕЛ")) {toUp.Add(tmp); break;}

//если в ячейке обьект И я пошел ЧЕРными И обьект\_ЧЕР, то несветим

if (figName.Contains("ЧЕР")) break;

}

if (!goBlack && figName != "null")

{

//если в ячейке обьект И я пошел БЕЛыми И обьект\_ЧЕР, то подсветим

if (figName.Contains("ЧЕР")) { toUp.Add(tmp); break; }

//если в ячейке обьект И я пошел БЕЛыми И обьект\_БЕЛ, то несветим

if (figName.Contains("БЕЛ")) break;

}

}

foreach (var bevel in \_bevels)

foreach (string s in toUp)

if (bevel.Tag.ToString() == s)

bevel.BackColor = Color.Green;

toUp.Clear();

#endregion

#region Нижнее направление

for (int i = (indexStart.X + 1); i <= down; i++)

{//6..5..4..0

string tmp = (i) + "" + indexStart.Y;

string figName = newGame.ShowFigName(tmp);

//если ячейка пуста, то подсветим:

if (figName == "null") toUp.Add(tmp);

if (goBlack && figName != "null")

{

//если в ячейке обьект И я пошел ЧЕРными И обьект\_БЕЛ, то подсветим

if (figName.Contains("БЕЛ")) { toUp.Add(tmp); break; }

//если в ячейке обьект И я пошел ЧЕРными И обьект\_ЧЕР, то несветим

if (figName.Contains("ЧЕР")) break;

}

if (!goBlack && figName != "null")

{

//если в ячейке обьект И я пошел БЕЛыми И обьект\_ЧЕР, то подсветим

if (figName.Contains("ЧЕР")) { toUp.Add(tmp); break; }

//если в ячейке обьект И я пошел БЕЛыми И обьект\_БЕЛ, то несветим

if (figName.Contains("БЕЛ")) break;

}

}

foreach (var bevel in \_bevels)

foreach (string s in toUp)

if (bevel.Tag.ToString() == s)

bevel.BackColor = Color.Green;

toUp.Clear();

#endregion

#region Левое направление

for (int i = (indexStart.Y - 1); i >= left; i--)

{//6..5..4..0

string tmp = indexStart.X + "" + (i);

string figName = newGame.ShowFigName(tmp);

//если ячейка пуста, то подсветим:

if (figName == "null") toUp.Add(tmp);

if (goBlack && figName != "null")

{

//если в ячейке обьект И я пошел ЧЕРными И обьект\_БЕЛ, то подсветим

if (figName.Contains("БЕЛ")) { toUp.Add(tmp); break; }

//если в ячейке обьект И я пошел ЧЕРными И обьект\_ЧЕР, то несветим

if (figName.Contains("ЧЕР")) break;

}

if (!goBlack && figName != "null")

{

//если в ячейке обьект И я пошел БЕЛыми И обьект\_ЧЕР, то подсветим

if (figName.Contains("ЧЕР")) { toUp.Add(tmp); break; }

//если в ячейке обьект И я пошел БЕЛыми И обьект\_БЕЛ, то несветим

if (figName.Contains("БЕЛ")) break;

}

}

foreach (var bevel in \_bevels)

foreach (string s in toUp)

if (bevel.Tag.ToString() == s)

bevel.BackColor = Color.Green;

toUp.Clear();

#endregion

#region Правое направление

for (int i = (indexStart.Y + 1); i <= right; i++)

{//6..5..4..0

string tmp = indexStart.X + "" + (i);

string figName = newGame.ShowFigName(tmp);

//если ячейка пуста, то подсветим:

if (figName == "null") toUp.Add(tmp);

if (goBlack && figName != "null")

{

//если в ячейке обьект И я пошел ЧЕРными И обьект\_БЕЛ, то подсветим

if (figName.Contains("БЕЛ")) { toUp.Add(tmp); break; }

//если в ячейке обьект И я пошел ЧЕРными И обьект\_ЧЕР, то несветим

if (figName.Contains("ЧЕР")) break;

}

if (!goBlack && figName != "null")

{

//если в ячейке обьект И я пошел БЕЛыми И обьект\_ЧЕР, то подсветим

if (figName.Contains("ЧЕР")) { toUp.Add(tmp); break; }

//если в ячейке обьект И я пошел БЕЛыми И обьект\_БЕЛ, то несветим

if (figName.Contains("БЕЛ")) break;

}

}

foreach (var bevel in \_bevels)

foreach (string s in toUp)

if (bevel.Tag.ToString() == s)

bevel.BackColor = Color.Green;

toUp.Clear();

#endregion

}

/// <summary>

/// Для пешек подсветка хода

/// </summary>

private void GoPawn(string btnFigureTag)

{

bool goBlack = btnFigureTag.Contains("ЧЕР");

//заряжаю тег кнопки "ЧЕР\_ПЕШКА\_A2", получаю индекс [6,0] в массиве

Point indexStart = newGame.PoiskVMassive(btnFigureTag);//[6,0]

if (indexStart.X < 0 || indexStart.Y < 0)

throw new Exception("Error");

const int up = 0, down = 7, left = 0, right = 7;

string index = "", figName = "";

//Черные?

if (goBlack)

{

if((indexStart.X - 1)>=up)

{

figName = newGame.ShowFigName(indexStart.X - 1 + "" + indexStart.Y);

index = indexStart.X - 1 + "" + indexStart.Y;

/\*Цикл выполнит 1 ход (true && false)в верх

\* И два хода при условии (true && true)\*/

int extWhile = 0;

do

{

//есть кто на позиции [X-1;Y]?

if (figName == "null")

{//нет\\светим\\проверим следующ панель

foreach (Bevel bevel in \_bevels)

if (bevel.Tag.ToString() == index)

{ bevel.BackColor = Color.Green; break; }

} if (figName != "null") break;

//сменим позицию и индекс:

/\*ПОВЕСИТЬ ПРЕВРАЩЕНИЕ В ДРУГУЮ ФИГУРУ\*/if ((indexStart.X - 2) < up) break;

figName = newGame.ShowFigName(indexStart.X - 2 + "" + indexStart.Y);

index = indexStart.X - 2 + "" + indexStart.Y;

extWhile++;

} while (extWhile != 2 && !newGame.OneGoFigure(indexStart));//Два хода разрешены?

}

//верх лево

index = (indexStart.X - 1) + "" + (indexStart.Y - 1);

if ((indexStart.X - 1)>=up && (indexStart.Y - 1)>=left)

{

figName = newGame.ShowFigName(index);

if (figName != null && figName.Contains("БЕЛ"))

{

foreach (Bevel bevel in \_bevels)

if (bevel.Tag.ToString() == index)

{ bevel.BackColor = Color.Green; break; }

}

}

//верх право

index = (indexStart.X - 1) + "" + (indexStart.Y + 1);

if ((indexStart.X - 1) >= up && (indexStart.Y + 1)<=right)

{

figName = newGame.ShowFigName(index);

if (figName != null && figName.Contains("БЕЛ"))

{

foreach (Bevel bevel in \_bevels)

if (bevel.Tag.ToString() == index)

{ bevel.BackColor = Color.Green; break; }

} return;

}

}

else//белые

{

if ((indexStart.X + 1)<=down)

{

figName = newGame.ShowFigName(indexStart.X + 1 + "" + indexStart.Y);

index = indexStart.X + 1 + "" + indexStart.Y;

/\*Цикл выполнит 1 ход (true && false)в низ

\* И два хода при условии (true && true)\*/

int extWhile = 0;

do

{

//есть кто на позиции [X+1;Y]?

if (figName == "null")

{//нет\\светим\\проверим следующ панель

foreach (Bevel bevel in \_bevels)

if (bevel.Tag.ToString() == index)

{ bevel.BackColor = Color.Green; break; }

} if (figName != "null") break;

//сменим позицию и индекс:

/\*ПОВЕСИТЬ ПРЕВРАЩЕНИЕ В ДРУГУЮ ФИГУРУ\*/if ((indexStart.X + 2)>down) break;

figName = newGame.ShowFigName(indexStart.X + 2 + "" + indexStart.Y);

index = indexStart.X + 2 + "" + indexStart.Y;

extWhile++;

} while (extWhile != 2 && !newGame.OneGoFigure(indexStart));//Два хода разрешены?

}

//низ лево

index = (indexStart.X + 1) + "" + (indexStart.Y - 1);

if((indexStart.X + 1)<=down && (indexStart.Y - 1)>=left)

{

figName = newGame.ShowFigName(index);

if (figName != null && figName.Contains("ЧЕР"))

{

foreach (Bevel bevel in \_bevels)

if (bevel.Tag.ToString() == index)

{bevel.BackColor = Color.Green;break;}

}

}

//низ право

index = (indexStart.X + 1) + "" + (indexStart.Y + 1);

if ((indexStart.X + 1)<=down && (indexStart.Y + 1)<=right)

{

figName = newGame.ShowFigName(index);

if (figName != null && figName.Contains("ЧЕР"))

{

foreach (Bevel bevel in \_bevels)

if (bevel.Tag.ToString() == index)

{bevel.BackColor = Color.Green;break;}

}return;

}

}

}

/// <summary>

/// Нажатие на КНОПКУ-фигуру

/// </summary>

private void tbn\_Click(object sender, EventArgs e) { xod\_Click(sender, e); }

private static ButtonBase \_btnBase;

/// <summary>

/// Куда собираемся перейти

/// </summary>

private void bevel\_Click(object sender, EventArgs e) { xod\_Click(sender, e); }

/// <summary>

/// Смена текущего положения ФОРМА и МАССИВ

/// </summary>

private void perexod(Bevel bvl)

{

//изменить \_btnBase.Tag и Image на выбранную фигуру

//на форме:

if(newGame.ShowFigName(bvl.Tag.ToString())=="null")

{

ChangeFigurePawnToN(bvl);

bvl.Controls.Add(\_btnBase);

}

//в массиве:

//ЧЕР\_ПЕШКА\_A2/в/40

newGame.PerestanovkaVMassive(\_btnBase.Tag.ToString(), bvl.Tag.ToString());

noCheked();

}

/// <summary>

/// Сменить пешку на выбранную - она выжила и дошла до другого края

/// </summary>

private void ChangeFigurePawnToN(Bevel bvl)

{

var objFigure = \_btnBase.Tag.ToString();//"ЧЕР\_ПЕШКА\_A2"

string black = objFigure.Contains("ЧЕР") ? "ЧЕР\_" : "БЕЛ\_";

Point indexPanel = newGame.coordinate(bvl.Tag.ToString());

//Это пешка? и панель [0, 0..7] || [7, 0..7])

if (objFigure.Contains("ПЕШКА") && (indexPanel.X == 0 || indexPanel.X == 7))

{//MessageBox.Show("Наш клиент");

//выбираем фигуру:

\_fTransformPawn.ShowDialog();

//найти кнопку и сменить тег с пешки на выбр фигуру:

foreach (var @base in \_buttons)

{

if (\_btnBase.Name == @base.Name)

{

//проверить уникальность этой фигуры

do

{//уже есть такой//переназначить Тег

\_fTransformPawn.UpIncrement();

@base.Tag = black + \_fTransformPawn.SelectedFigure;//"ЧЕР\_КОНЬ\_1";

} while (Search());

break;

}

}

\_btnBase.Tag = black + \_fTransformPawn.SelectedFigure;//"ЧЕР\_КОНЬ\_1";

\_btnBase.Image = ChangeImageBtn(\_btnBase.Tag.ToString());

newGame.TransformPawn(objFigure, \_btnBase.Tag.ToString(), indexPanel);

}

}

/// <summary>

/// Вернет TRUE если такая фигура уже есть

/// </summary>

private bool Search()

{

string[] Massiv = new string[\_buttons.Count];

int i = 0, j = 0;

foreach (var ts in \_buttons)

{

Massiv[j] = ts.Tag.ToString();

j++;

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

if (ts.Tag.ToString().CompareTo(Massiv[e]) == 0)

{//Если есть результ, то есть совпадение с текущим значением ts!

return true;

}

i++;

}

return false;

}

/// <summary>

/// Сменит у кнопки рисунок пешки на другой выбранный

/// </summary>

private Image ChangeImageBtn(string name)

{

//проверить какой пакет установлен на фигурки в меню!

string[] tmp = name.Split('\_');

if(tmp[0].Contains("ЧЕР"))

{

switch (tmp[1])

{

case "btnILaden": return \_formMenu.SearchAndTransform("чер\_ладья\_nn", \_StateSelectImgPak);//return Resources.черный\_ладья;

case "btnIKoni": return \_formMenu.SearchAndTransform("чер\_конь\_nn", \_StateSelectImgPak);//return Resources.черный\_конь;

case "btnISlon": return \_formMenu.SearchAndTransform("чер\_слон\_nn", \_StateSelectImgPak);//return Resources.черный\_слон;

case "btnIFerz": return \_formMenu.SearchAndTransform("чер\_ферзь\_nn", \_StateSelectImgPak);//return Resources.черный\_ферзь;

}

}

else

{

switch (tmp[1])

{

case "btnILaden": return \_formMenu.SearchAndTransform("бел\_ладья\_nn", \_StateSelectImgPak);//return Resources.белый\_ладья;

case "btnIKoni": return \_formMenu.SearchAndTransform("бел\_конь\_nn", \_StateSelectImgPak);//return Resources.белый\_конь;

case "btnISlon": return \_formMenu.SearchAndTransform("бел\_слон\_nn", \_StateSelectImgPak);//return Resources.белый\_слон;

case "btnIFerz": return \_formMenu.SearchAndTransform("бел\_ферзь\_nn", \_StateSelectImgPak);//return Resources.белый\_ферзь;

}

}

return null;

}

//private List<ButtonBase> reBtn = new List<ButtonBase>();

/// <summary>

/// Удалить фигуру на форме

/// </summary>

private void deliteFigure(ButtonBase когоСнять, string когоПоставить)

{

//заряжаю тег кнопки "ЧЕР\_СЛОН\_A2", получаю индекс [6,0] в массиве

Point indexStart = newGame.PoiskVMassive(когоСнять.Tag.ToString()); //[6,0]

//удалить и в массиве

newGame.DeliteFigure(когоСнять.Tag.ToString(), когоПоставить);

//удалить на форме (сделать неактивным-невидимым)

когоСнять.Visible = false;

когоСнять.Enabled = false;

//reBtn.Add(когоСнять);

//ФОРМА: поставить новую фигуру на место-панель битой

foreach (Bevel bevel in \_bevels.Where(bevel => bevel.Tag.ToString() == (indexStart.X + "" + indexStart.Y)))

{

foreach (ButtonBase @base in \_buttons.Where(@base => @base.Tag.ToString() == когоПоставить))

{

ChangeFigurePawnToN(bevel);

bevel.Controls.Add(@base);

break;

}

break;

}

noCheked();

if(когоСнять.Tag.ToString().Contains("КАРОЛЬ"))

{

if (когоСнять.Tag.ToString().Contains("БЕЛ"))

\_saveToFile.BlackWin += 1;

MessageBox.Show("КОНЕЦ ИГРЫ");

//Вызвать статистику

настройкиToolStripMenuItem\_Click(null, null);

}

}

/// <summary>

/// Восстановить поле после подсветки хода

/// </summary>

private void noCheked()

{

foreach (var bevel in \_bevels.Where(bevel => bevel != null))

{

bevel.BackColor = Color.Transparent;

}

}

/// <summary>

/// Начать игру заново.

/// начало игры на чистом поле, сброс предыдущей игры;

/// </summary>

private void newGameToolStripMenuItem\_Click(object sender, EventArgs e)

{

//сбросить поле с фигурами

newGame = new clNewGame();

\_saveToFile.Schetchik += 1;

//сбросить очередность первого хода

if (\_formMenu.IsEvent()) xod.BlackGo = \_formMenu.OneGo;

xod = new clXod(\_formMenu.OneGo);

//сбросить кнопки на форме

ResetForm();

}

/// <summary>

/// Сбросить все кнопки на исходные места

/// </summary>

private void ResetForm()

{

this.bevel\_A1.Controls.Add(this.btnBL\_a1); this.bevel\_B2.Controls.Add(this.btnBP\_b2);

this.bevel\_C1.Controls.Add(this.btnBS\_c1); this.bevel\_D2.Controls.Add(this.btnBP\_d2);

this.bevel\_E1.Controls.Add(this.btnBF\_e1); this.bevel\_G1.Controls.Add(this.btnBK\_g1);

this.bevel\_H2.Controls.Add(this.btnBP\_h2); this.bevel\_F2.Controls.Add(this.btnBP\_f2);

this.bevel\_G2.Controls.Add(this.btnBP\_g2); this.bevel\_E2.Controls.Add(this.btnBP\_e2);

this.bevel\_C2.Controls.Add(this.btnBP\_c2); this.bevel\_A2.Controls.Add(this.btnBP\_a2);

this.bevel\_B1.Controls.Add(this.btnBK\_b1); this.bevel\_D1.Controls.Add(this.btnBW\_d1);

this.bevel\_F1.Controls.Add(this.btnBS\_f1); this.bevel\_H1.Controls.Add(this.btnBL\_h1);

this.bevel\_A7.Controls.Add(this.btnWP\_a7); this.bevel\_B7.Controls.Add(this.btnWP\_b7);

this.bevel\_C7.Controls.Add(this.btnWP\_c7); this.bevel\_D7.Controls.Add(this.btnWP\_d7);

this.bevel\_F7.Controls.Add(this.btnWP\_f7); this.bevel\_E7.Controls.Add(this.btnWP\_e7);

this.bevel\_H7.Controls.Add(this.btnWP\_h7); this.bevel\_G7.Controls.Add(this.btnWP\_g7);

this.bevel\_H8.Controls.Add(this.btnWL\_h8); this.bevel\_G8.Controls.Add(this.btnWK\_g8);

this.bevel\_F8.Controls.Add(this.btnWS\_f8); this.bevel\_E8.Controls.Add(this.btnWF\_e8);

this.bevel\_D8.Controls.Add(this.btnWW\_d8); this.bevel\_C8.Controls.Add(this.btnWS\_c8);

this.bevel\_A8.Controls.Add(this.btnWL\_a8); this.bevel\_B8.Controls.Add(this.btnWK\_b8);

foreach (var button in \_buttons)

{

foreach (var btnDeff in \_copyValueObj.ObjValue)

{

if (button.Name == btnDeff.Key && button.Tag.ToString() != btnDeff.Value && btnDeff.Value != null)

{

button.Tag = btnDeff.Value;

button.Image = button.Tag.ToString().Contains("ЧЕР")

? Resources.черный\_пешка : Resources.белый\_пешка;

}

}

button.Enabled = true;

button.Visible = true;

}

}

/// <summary>

/// Выход из программы

/// </summary>

private void exitToolStripMenuItem\_Click(object sender, EventArgs e) { this.Close(); }

/// <summary>

/// сохранение текущей игры в любой момент в файл;

/// </summary>

private void saveToolStripMenuItem\_Click(object sender, EventArgs e)

{

//сохранить содержимое массива в файл

//сохранить Кто\_ходил\_последним

var saveFileDialog = new SaveFileDialog {Filter = "Текстовые файлы|\*.txt|Все файлы|\*.\*"};

if (saveFileDialog.ShowDialog() != DialogResult.OK)

return;

var saveData = new List<string>();

saveData.AddRange(newGame.CopyFiguresDataToList());

var streamWriter = new StreamWriter(saveFileDialog.FileName);

streamWriter.WriteLine(xod.BlackGo);

foreach (string s in saveData)

streamWriter.WriteLine(s);

streamWriter.Close();

}

/// <summary>

/// восстановление состояния игры из файла;

/// </summary>

private void loadFromFileToolStripMenuItem\_Click(object sender, EventArgs e)

{

OpenFileDialog openFileDialog = new OpenFileDialog {Filter = "Текстовые файлы|\*.txt|Все файлы|\*.\*"};

if(openFileDialog.ShowDialog()!=DialogResult.OK)

return;

List<string > loadData = new List<string>();

StreamReader streamReader = new StreamReader(openFileDialog.FileName);

//переписать чей Первый\_Ход после загрузки

xod.BlackGo = streamReader.ReadLine().Contains("False")?false:true;

while (true)

{

string str = streamReader.ReadLine();

if(str==null)break;

loadData.Add(str);

}

streamReader.Close();

//загрузить из файла массив и поместить его в игру.

if (loadData.Count > 3) newGame.PastDataListToFiguresArr(loadData);

else

{

MessageBox.Show("Файл повреджен, либо не содержит загрузочных данных.", "Ошибка", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

//расставить фигуры-кнопки на доске ориентируясь только на

//содержимое загруженого массива

//на форме:

RasstanovkaKnopok();

}

/// <summary>

/// Показать подсказку

/// </summary>

private void showBall(string typeFigure)

{

if(\_formMenu.IsShowHelp())

{

Balloon b = new Balloon

{

Style = eBallonStyle.Alert,

CaptionText = "Недопустимый ход"

};

b.MouseHover += new EventHandler(ballon\_MouseHover);

b.MouseLeave += new EventHandler(ballon\_MouseLeave);

switch (typeFigure)

{//вызвать как ходит фигура

case "ПЕШКА":

{

b.CaptionImage = Resources.pawn128;

b.Text = Resources.fMain\_showBall\_Пешка;

} break;

case "ЛАДЬЯ":

{

b.CaptionImage = Resources.rook26;

b.Text = Resources.fMain\_showBall\_Ладья;

} break;

case "КОНЬ":

{

b.CaptionImage = Resources.knight48;

b.Text = Resources.fMain\_showBall\_Конь;

} break;

case "СЛОН":

{

b.CaptionImage = Resources.bishop48;

b.Text = Resources.fMain\_showBall\_Слон;

} break;

case "КАРОЛЬ":

{

b.CaptionImage = Resources.king26;

b.Text = Resources.fMain\_showBall\_Кароль;

} break;

case "ФЕРЗЬ":

{

b.CaptionImage = Resources.queen48;

b.Text = Resources.fMain\_showBall\_Ферзь;

} break;

//вызвать предупреждение

case "errBlackGo":

{

b.CaptionImage = Resources.alert;

b.Text = Resources.fMain\_showBall\_BlackGo;

} break;

case "errWhiteGo":

{

b.CaptionImage = Resources.alert;

b.Text = Resources.fMain\_showBall\_WhiteGo;

} break;

case "errSHAX":

{

b.CaptionImage = Resources.alert;

b.Text = Resources.fMain\_showBall\_SHAX;

} break;

case "errNe":

{

b.CaptionImage = Resources.alert;

b.Text = Resources.fMain\_showBall\_Ne;

} break;

default: return;

}

b.AlertAnimation = eAlertAnimation.TopToBottom;

b.AutoResize();

if(\_flagCliseBall)

{

b.AutoClose = true;

b.AutoCloseTimeOut = 3;

}

b.Owner = this;

b.Show(menuStrip1, false);

}

}

private int \_StateSelectImgPak = 0;

private fMenu \_formMenu = new fMenu();

/// <summary>

/// Открыть окно с настройками

/// </summary>

private void настройкиToolStripMenuItem\_Click(object sender, EventArgs e)

{

//статистика обновить

\_formMenu.Statistic(\_saveToFile.Schetchik);

//количество выйгрышей обновить

\_formMenu.UpdateLiveChart(\_saveToFile.BlackWin);

\_formMenu.ShowDialog();

//Если текущий неравен предыдущему, то стиль меняем

if (\_formMenu.StateSelectImgPak != \_StateSelectImgPak)

ChangeFiguresStylePak();

if (\_formMenu.BoardImage != null && \_formMenu.BoardImage != tableLayoutPanel1.BackgroundImage)

tableLayoutPanel1.BackgroundImage = \_formMenu.BoardImage;

//очередность хода:

if(\_formMenu.IsEvent()) xod.BlackGo = \_formMenu.OneGo;

if(\_formMenu.IsNewKeys)

{

\_кнопкиУправления.WriteParam(\_formMenu.\_arr);

\_formMenu.IsNewKeys = false;

NauStateKey();

}

}

/// <summary>

/// Изменить Image у всех кнопок

/// </summary>

private void ChangeFiguresStylePak()

{

//перепишем текущий стиль для будущего сравнения

\_StateSelectImgPak = \_formMenu.StateSelectImgPak;

foreach (var button in \_buttons)

button.Image = \_formMenu.SearchAndTransform(button.Tag.ToString(), \_StateSelectImgPak);

}

/// <summary>

/// Чем рулить. Либо дифолтом - с файла, либо назнач в настройках

/// </summary>

KeyControlsDeff \_кнопкиУправления = new KeyControlsDeff();

private Bevel \_bvlObj;

/// <summary>

/// Из чего нажато - отсеиваем кнопы управления(заданы\умолч.) и меняем bevel.border

/// </summary>

private void fMain\_KeyUp(object sender, KeyEventArgs e)

{

var key = e.KeyCode;

switch (key)

{

case Keys.Enter://генерим мыху xod\_Click(object sender, EventArgs e)

//sender as button || bevel//e as Point[x,y]

if(\_bvlObj!=null) xod\_Click(UseEnter(), null);

break;

}

//Ориентируемся по кодам клавиш

foreach (KeyValuePair<string, string > s in \_keysB)

{

if (e.KeyValue.ToString() == s.Value)

{

//Реакция панели на нажатие совпавшей клавиши:

ChangeBevelBorder(s);

return;

}

}

foreach (KeyValuePair<string,string > s in \_keysW)

{

if (e.KeyValue.ToString() == s.Value)

{

//Реакция панели на нажатие совпавшей клавиши:

ChangeBevelBorder(s);

return;

}

}

}

/// <summary>

/// Вернуть object либо Bevel либо ButtonBase

/// </summary>

private object UseEnter()

{

//Если на панели есть обьект то вернуть обьект

string objBtn = newGame.ShowFigName(\_bvlObj.Tag.ToString());

if (objBtn == "null") return \_bvlObj;//Иначе вернуть панель

return \_buttons.FirstOrDefault(@base => @base.Tag.ToString() == objBtn);

}

private bool \_chColor;

private string \_bevelStateIndex;

/// <summary>

/// Изменить Визуально активную панель взависимости от нажатой клавы

/// </summary>

private void ChangeBevelBorder(KeyValuePair<string,string > keyObject)

{

//keyObject даст пару имя.значение - тоесть куда идти и нажата кнопка

//Сбросить предыдущий стиль панели

foreach (var bevel in \_bevels.Where(bevel => bevel.Tag.ToString() == \_bevelStateIndex))

{

bevel.BackColor = \_chColor ? Color.Green : Color.Transparent;

break;

}

\_chColor = false;

if (\_кнопкиУправления.Go(keyObject.Key, \_bevelStateIndex) == null) { \_bvlObj = null; return; }

\_bevelStateIndex = \_кнопкиУправления.Go(keyObject.Key, \_bevelStateIndex);

//найти нужную панель и сменить оформление

//ориентир поиска парам Name из XML файла (up\down\left\right)

//за счет этого меняем на 1 индекс движения

//44-->up-->34

//44-->down-->54

foreach (var bevel in \_bevels.Where(bevel => bevel.Tag.ToString() == \_bevelStateIndex))

{

if (bevel.BackColor == Color.Green) \_chColor=true;

bevel.BackColor = Color.Cornsilk;

\_bvlObj = bevel;

bevel.Focus();

return;

}

\_bvlObj = null;

}

/// <summary>

/// Перед закрытием окна делаем сохранение в файл

/// </summary>

private void fMain\_FormClosing(object sender, FormClosingEventArgs e)

{

//сохранить счетчик в файл

\_saveToFile.Save();

}

/// <summary>

/// Проверить возможность ШАХа фигурой

/// </summary>

private void ProverkaNaShax(string btnTagFigure)

{

//берем фигуру после действия ход\битьё

//берем "ЧЕР" или "БЕЛ"

if(\_formMenu.ShaxOn)

{

if(btnTagFigure.Contains("ЧЕР"))

{//проверить Угроза Белому королю

//берем положение короля БЕЛ в массиве

Point indexKing = newGame.PoiskVMassive("БЕЛ\_КАРОЛЬ\_D8");

//могу ли фигурой дойти до панели с королем?

switch (newGame.VariantXoda(btnTagFigure, (indexKing.X + "" + indexKing.Y), false))

{

case "OK": showBall("errSHAX"); break;

case "Error": newGameToolStripMenuItem\_Click(null, null); break;

}

}

else

{//проверить Угроза Черному королю

//берем положение короля ЧЕР в массиве

Point indexKing = newGame.PoiskVMassive("ЧЕР\_КАРОЛЬ\_D1");

//могу ли фигурой дойти до панели с королем?

switch (newGame.VariantXoda(btnTagFigure, (indexKing.X + "" + indexKing.Y), false))

{

case "OK": showBall("errSHAX"); break;

case "Error": newGameToolStripMenuItem\_Click(null, null); break;

}

}

}

}

private bool \_flagCliseBall = true;

/// <summary>

/// Курсор над обьектом Подсказка

/// </summary>

private void ballon\_MouseHover(object sender, EventArgs e)

{

var obj = (sender as Balloon);

if (obj != null)

{

obj.AutoCloseTimeOut += 5;

}

}

/// <summary>

/// Курсор покинул обьект Подсказка

/// </summary>

private void ballon\_MouseLeave(object sender, EventArgs e)

{

var obj = (sender as Balloon);

if (obj != null)

{

obj.AutoClose = true;

obj.AutoCloseTimeOut = 1;

\_flagCliseBall = true;

}

}

/// <summary>

/// Установить кнопки на форме так как они есть в массиве

/// </summary>

private void RasstanovkaKnopok()

{

foreach (var button in \_buttons)

{

button.Visible = false;

button.Enabled = false;

}

//проход по панелям

foreach (var bevel in \_bevels)

{

string objFigure = newGame.ShowFigName(bevel.Tag.ToString());

//если в массиве по адресу Тег\_панели есть обьект

//то на панель ставим кнопку с Тегом\_имени\_обьекта

//иначе пусто

if(objFigure!="null")

foreach (var button in \_buttons.Where(button => button.Tag.ToString() == objFigure))

{

button.Enabled = true;

button.Visible = true;

bevel.Controls.Add(button);

}

}

ChangeFigurePawnToN();

}

/// <summary>

/// Сменить тег кнопки на измененый

/// </summary>

private void ChangeFigurePawnToN()

{

foreach (var bevel in \_bevels)

{

string objFigure = newGame.ShowFigName(bevel.Tag.ToString());

string[] tmp = objFigure.Split('\_');

//если в массиве по адресу Тег\_панели есть обьект

//то на панель ставим кнопку с Тегом\_имени\_обьекта

//иначе пусто

if (objFigure != "null")

foreach (var button in \_buttons)

{

if (!button.Enabled && button.Tag.ToString().Contains("ПЕШКА") && tmp != null)

switch (tmp[1])

{

case "btnILaden":

button.Enabled = true;

button.Visible = true;

button.Tag = objFigure;

button.Image = ChangeImageBtn(objFigure);

bevel.Controls.Add(button);

tmp = null;

break;

case "btnIKoni":

button.Enabled = true;

button.Visible = true;

button.Tag = objFigure;

button.Image = ChangeImageBtn(objFigure);

bevel.Controls.Add(button);

tmp = null;

break;

case "btnISlon":

button.Enabled = true;

button.Visible = true;

button.Tag = objFigure;

button.Image = ChangeImageBtn(objFigure);

bevel.Controls.Add(button);

tmp = null;

break;

case "btnIFerz":

button.Enabled = true;

button.Visible = true;

button.Tag = objFigure;

button.Image = ChangeImageBtn(objFigure);

bevel.Controls.Add(button);

tmp = null;

break;

}

}

}

//если обьект есть в массиве, а нет кнопки с таким тегом)

//выбрать неактивную кнопку с тегом ПЕШКА

//преобразовать ее явно в "ibtn"

}

/// <summary>

/// (-1) - текущее\_состояние.

/// (-2) - влево на 1 от текущего

/// </summary>

private int \_stackIndexer, \_i = 1;

private bool \_lastflag=true;

/// <summary>

/// Сделать Откат назад/вперед

/// </summary>

private void Otkat\_Click(object sender, EventArgs e)

{

if (\_i <= 0) \_i = 1;

if (((ToolStripMenuItem)sender).Name == goBack.Name && \_stackXodov.GetStringListObject(\_stackXodov.CountIndex-1) != null)

{

if(\_i==1 && \_lastflag)

{

\_i = \_stackXodov.CountIndex - 2;

\_stackIndexer = \_i;

\_lastflag = false;

}

//при откате назат на 1 берем -1\_последний элемент стека //и переписываем его в конец стека.

\_stackXodov.AddString(\_stackXodov.GetStringListObject(\_i), \_stackXodov.GetBoolObject(\_i));//\_stackXodov.Add(\_stackXodov.GetIndexObject(\_stackIndexer), \_stackXodov.GetBoolObject(\_stackIndexer));

//Загружаем игру с конца стека

NState();

\_i--;

return;

}

//else if (\_stackXodov.GetStringListObject(2) != null) \_stackIndexer++;

if ((sender as ToolStripMenuItem).Name == goForward.Name && \_stackXodov.GetStringListObject(\_stackXodov.CountIndex - 1) != null)

{

if (!\_lastflag)

{

\_i += 2;

\_lastflag = true;

}else if(\_stackIndexer+1 > \_i) \_i++;

//при откате назат на 1 берем -1\_последний элемент стека //и переписываем его в конец стека.

\_stackXodov.AddString(\_stackXodov.GetStringListObject(\_i), \_stackXodov.GetBoolObject(\_i));//\_stackXodov.Add(\_stackXodov.GetIndexObject(\_stackIndexer), \_stackXodov.GetBoolObject(\_stackIndexer));

//Загружаем игру с конца стека

NState();

}

}

/// <summary>

/// Восстановить игру из отката

/// </summary>

private void NState()

{

//загрузить из файла массив и поместить его в игру.

if (\_stackXodov.GetLastListObject() != null)

{

//переписать чей Первый\_Ход после загрузки

xod.BlackGo = \_stackXodov.GetLastBoolObject();

//newGame.PastDataListToFiguresArr(\_stackXodov.GetLastObject());

newGame.PastDataListToFiguresArr(\_stackXodov.GetLastListObject());

//расставить фигуры-кнопки на доске ориентируясь только на

//содержимое загруженого массива

//на форме:

RasstanovkaKnopok();

}

}

}

}

* 1. **Класс clNewGame – движок игры, внутренняя логика ходов:**

using System;

using System.Collections.Generic;

using System.Drawing;

namespace Шахматы

{

class clNewGame

{

private clFigure[,] figure = new clFigure[8,8];

public clNewGame()

{

Restart();

}

/\*

\_\_0\_1\_2\_3\_4\_5\_6\_7\_\_

0|1 2 3 5 4 3 2 1|8

1|0 0 0 0 0 0 0 0|7

2| |6

3| |5

4| |4

5| |3

6|0 0 0 0 0 0 0 0|2

7|1 2 3 5 4 3 2 1|1

\_\_a\_b\_c\_d\_e\_f\_g\_h\_\_

\*/

/// <summary>

/// Сбросить на начальные позиции фигуры в МАССИВЕ

/// </summary>

private void Restart()

{//СТРОКА\\СТОЛБЕЦ

var WP10 = new clFigure("БЕЛ\_ПЕШКА\_A7"); var WP11 = new clFigure("БЕЛ\_ПЕШКА\_B7");

var WP12 = new clFigure("БЕЛ\_ПЕШКА\_C7"); var WP13 = new clFigure("БЕЛ\_ПЕШКА\_D7");

var WP14 = new clFigure("БЕЛ\_ПЕШКА\_E7"); var WP15 = new clFigure("БЕЛ\_ПЕШКА\_F7");

var WP16 = new clFigure("БЕЛ\_ПЕШКА\_G7"); var WP17 = new clFigure("БЕЛ\_ПЕШКА\_H7");

var WL00 = new clFigure("БЕЛ\_ЛАДЬЯ\_A8"); var WL07 = new clFigure("БЕЛ\_ЛАДЬЯ\_H8");

var WK01 = new clFigure("БЕЛ\_КОНЬ\_B8"); var WK06 = new clFigure("БЕЛ\_КОНЬ\_G8");

var WS02 = new clFigure("БЕЛ\_СЛОН\_C8"); var WS05 = new clFigure("БЕЛ\_СЛОН\_F8");

var WW03 = new clFigure("БЕЛ\_КАРОЛЬ\_D8");var WF04 = new clFigure("БЕЛ\_ФЕРЗЬ\_E8");

var BP60 = new clFigure("ЧЕР\_ПЕШКА\_A2"); var BP61 = new clFigure("ЧЕР\_ПЕШКА\_B2");

var BP62 = new clFigure("ЧЕР\_ПЕШКА\_C2"); var BP63 = new clFigure("ЧЕР\_ПЕШКА\_D2");

var BP64 = new clFigure("ЧЕР\_ПЕШКА\_E2"); var BP65 = new clFigure("ЧЕР\_ПЕШКА\_F2");

var BP66 = new clFigure("ЧЕР\_ПЕШКА\_G2"); var BP67 = new clFigure("ЧЕР\_ПЕШКА\_H2");

var BL70 = new clFigure("ЧЕР\_ЛАДЬЯ\_A1"); var BL77 = new clFigure("ЧЕР\_ЛАДЬЯ\_H1");

var BK71 = new clFigure("ЧЕР\_КОНЬ\_B1"); var BK76 = new clFigure("ЧЕР\_КОНЬ\_G1");

var BS72 = new clFigure("ЧЕР\_СЛОН\_C1"); var BS75 = new clFigure("ЧЕР\_СЛОН\_F1");

var BW73 = new clFigure("ЧЕР\_КАРОЛЬ\_D1");var BF74 = new clFigure("ЧЕР\_ФЕРЗЬ\_E1");

figure[0, 0] = WL00; figure[0, 1] = WK01; figure[0, 2] = WS02; figure[0, 3] = WW03;

figure[0, 7] = WL07; figure[0, 6] = WK06; figure[0, 5] = WS05; figure[0, 4] = WF04;

//пешки

figure[1, 0] = WP10; figure[1, 1] = WP11; figure[1, 2] = WP12; figure[1, 3] = WP13;

figure[1, 4] = WP14; figure[1, 5] = WP15; figure[1, 6] = WP16; figure[1, 7] = WP17;

//пешки

figure[6, 0] = BP60; figure[6, 1] = BP61; figure[6, 2] = BP62; figure[6, 3] = BP63;

figure[6, 4] = BP64; figure[6, 5] = BP65; figure[6, 6] = BP66; figure[6, 7] = BP67;

figure[7, 0] = BL70; figure[7, 1] = BK71; figure[7, 2] = BS72; figure[7, 3] = BW73;

figure[7, 7] = BL77; figure[7, 6] = BK76; figure[7, 5] = BS75; figure[7, 4] = BF74;

}

/// <summary>

/// Какая фигура\нет по адресу [0,A]

/// </summary>

public string ShowFigName(int str, int stolb)

{

//MessageBox.Show(BtnClik);

var obj = figure[str, stolb];

return obj.Name;

}

/// <summary>

/// Какая фигура\нет по тегу панели [40]

/// </summary>

public string ShowFigName(string nn)

{

if (nn == "" || nn.Contains("-")) return "null";

string str = nn.Substring(0, 1);

string stolb = nn.Substring(1, 1);

int sr = Convert.ToInt32(str);

int sb = Convert.ToInt32(stolb);

var obj = figure[sr, sb];

if(obj!=null)return obj.Name;

return "null";

}

/// <summary>

/// Перегнать строкy "40" в цифры Point[4, 0]

/// </summary>

internal Point coordinate(string pnl)

{

if(pnl.Contains("-"))return new Point(-1,-1);

string str = pnl.Substring(0, 1);

string stolb = pnl.Substring(1, 1);

return new Point(Convert.ToInt32(str), Convert.ToInt32(stolb));

}

/// <summary>

/// Перенести фигуру с ячейки [btn] в [panel]

/// </summary>

public void PerestanovkaVMassive(string btnStartTag, string panelEndTag)

{

Point end = coordinate(panelEndTag);

//взять из ячейки

for (int i = 0; i < 8; i++)//проход по строке

{

for (int j = 0; j < 8; j++ )//проход по ячейкам строки

if (figure[i, j] != null && figure[i, j].Name == btnStartTag)

{

var tmp = figure[i, j];

//если фигура ПЕШКА то можно разок сходить на два++

if(btnStartTag.Contains("ПЕШКА")) tmp.OneGo = true;

//затереть ячейку старта

figure[i, j] = null;

//положить в новую ячейку

figure[end.X, end.Y] = tmp;

break;

}

}

}

/// <summary>

/// Удалить обьект в массиве

/// </summary>

public void DeliteFigure(string btnTagOld, string btnTagNew)

{

var temp = new clFigure("");

for (int i = 0; i < 8; i++)//проход по строке

{

for (int j = 0; j < 8; j++)//проход по ячейкам строки

if (figure[i, j] != null && figure[i, j].Name == btnTagNew)

{

temp = figure[i, j];

//затереть ячейку старта

figure[i, j] = null;

break;

}

}

for (int i = 0; i < 8; i++)//проход по строке

{

for (int j = 0; j < 8; j++)//проход по ячейкам строки

if (figure[i, j] != null && figure[i, j].Name == btnTagOld)

{

figure[i, j] = temp;

break;

}

}

}

/// <summary>

/// Найти обьект-кнопку в массиве по известному полю Tag

/// Вернет индекс обьекта

/// </summary>

public Point PoiskVMassive(string btnStartTag)

{

//взять из ячейки

for (var i = 0; i < 8; i++)//проход по строке

{

for (var j = 0; j < 8; j++)//проход по ячейкам строки

if (figure[i, j] != null && figure[i, j].Name == btnStartTag)

{

return new Point(i, j);

}

}

return new Point(-1, -1);

}

/// <summary>

/// Изменить поле Имя у обьекта ПЕШКА в массиве

/// </summary>

public void TransformPawn(string figOld, string figNew, Point indexPanel)

{

//внести новый обьект transformPawn

figure[indexPanel.X, indexPanel.Y] = new clFigure(figNew);

//занулить старый обьект ПЕШКА

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

{

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

{

if(figure[i,j] != null && figure[i,j].Name==figOld)

{

figure[i, j] = null;

break;

}

}

}

}

/// <summary>

/// Кто как может пойти.

/// Заряжаю btnTag кнопки "ЧЕР\_КАРОЛЬ\_A2", panelEndTag "40"

/// </summary>

/// <param name="pawnBatle">true - пешка бьет</param>

public string VariantXoda(string btnTag, string panelEndTag, bool pawnBatle)

{

//если кнопке xod.XodStartFigureTag разрешен переход на панель bvl.Tag:

//если на панели есть фигура, то бить ИЛИ (это своя, туда нельзя)

//FALSE если на панели фигура\_своих, если панель под кнопкой

//TRUE если на панели нет\_фигур, если на панели фигура противника

//цвет фигуры

//bool white = btnTag.Contains("БЕЛ");

bool black = btnTag.Contains("ЧЕР");

//определить - чё за кнопка?//тип фигуры

int scase = -1;

if (btnTag.Contains("ПЕШКА")) scase = 0;

else if (btnTag.Contains("ЛАДЬЯ") || btnTag.Contains("ILaden")) scase = 1;

else if (btnTag.Contains("КОНЬ") || btnTag.Contains("IKoni")) scase = 2;

else if (btnTag.Contains("СЛОН") || btnTag.Contains("ISlon")) scase = 3;

else if (btnTag.Contains("КАРОЛЬ")) scase = 4;

else if (btnTag.Contains("ФЕРЗЬ") || btnTag.Contains("IFerz")) scase = 5;

//индексы Фигура-->Клетка

Point end = coordinate(panelEndTag);//"40"->[4,0]

if (end.X < 0 || end.Y < 0) return "Error";

Point start = PoiskVMassive(btnTag);//"ЧЕР\_КАРОЛЬ\_A2"->[1,0]

//Разные методики хода:

switch (scase)

{

case 0://пешка

{

//Могу ли 'пешка' дойти до панели end

//И нет ли фигур на всем пути исключая саму панель

string result = GoPawn(start, end, black ? true : false, pawnBatle);

if (result == "help" || result == "Путь перекрыт" || result == "err" || result == "OK")

return result;

}

break;

case 1://Ладья

{

string result = GoLaden(start, end, black ? true : false);

if (result == "help" || result == "Путь перекрыт" || result == "err" || result == "OK")

return result;

}

break;

case 2://Конь

{

string result = GoKoni(start, end, black ? true : false);

if (result == "help" || result == "Путь перекрыт" || result == "err" || result == "OK")

return result;

}

break;

case 3://Слон

{

string result = GoSlon(start, end, black ? true : false, true);

if (result == "help" || result == "Путь перекрыт" || result == "err" || result == "OK")

return result;

}

break;

case 4://Кароль

{

string result = GoKarol(start, end, black ? true : false);

if (result == "help" || result == "Путь перекрыт" || result == "err" || result == "OK")

return result;

}

break;

case 5://Ферзь

{

string resultS = GoSlon(start, end, black ? true : false, false);

string resultL = GoLaden(start, end, black ? true : false);

if ((resultL == "help" && resultS == "help"))

return resultL;

if ((resultL == "Путь перекрыт" && resultS == "Путь перекрыт"))

return resultL;

if ((resultL == "Путь перекрыт" && resultS == "help"))

return resultL;

if ((resultL == "help" && resultS == "Путь перекрыт"))

return resultS;

if ((resultL == "help" && resultS == "OK"))

return resultS;

if ((resultL == "OK" && resultS == "help"))

return resultL;

if ((resultL == "OK" && resultS == "OK"))

{/\*-->"Есть ли обьект на панели?" \*/

//MessageBox.Show("Че на самой панели?");

return resultL;

}

}

break;

}

return "";

}

/// <summary>

/// Для короля методика хода.

/// Могу ли королем дойти до панели end?

/// Контролировать расстояние в 1 клетку

/// </summary>

private string GoKarol(Point indexStart, Point end, bool black)

{

//набор тестов на равенство:

//а) середина И (лево ИЛИ право)

//б) выше И (лево ИЛИ право ИЛИ центр)

//в) ниже И (лево ИЛИ право ИЛИ центр)

if (indexStart.X == end.X && (indexStart.Y - 1 == end.Y || indexStart.Y + 1 == end.Y))

return "OK";

if (indexStart.X - 1 == end.X && (indexStart.Y - 1 == end.Y || indexStart.Y + 1 == end.Y || indexStart.Y == end.Y))

return "OK";

if (indexStart.X + 1 == end.X && (indexStart.Y - 1 == end.Y || indexStart.Y + 1 == end.Y || indexStart.Y == end.Y))

return "OK";

return "help";

}

/// <summary>

/// Для слона методика хода.

/// Могу ли сланом дойти до панели end?

/// И нет ли фигур на всем пути исключая саму панель.

/// </summary>

private string GoSlon(Point indexStart, Point end, bool black, bool goSlon)

{

bool onDiagonal = false;

//если это слон а не ферзь:

if (goSlon && (indexStart.X == end.X || indexStart.Y == end.Y)) return "help";

//панель выше?

if (indexStart.X > end.X)

{//панель левее?

if(indexStart.Y > end.Y)

{//идти диагонально по 1 клетке //подтвердить что панель лежит на диагонали

for (int str = indexStart.X-1, stolb = indexStart.Y-1; str >= end.X; str--, stolb--)

if (str == end.X && stolb == end.Y) { onDiagonal = true; break; }

//на диагонали

if (onDiagonal)

{//дойти до панели диагональю

for (int str = indexStart.X-1, stolb = indexStart.Y-1; str > end.X; str--, stolb--)

{//искать другие фигуры на пути

if (figure[str, stolb] != null) return "Путь перекрыт";

} return "OK";

} return "help";

}

//правее//идти диагонально по 1 клетке //подтвердить что панель лежит на диагонали

for (int str = indexStart.X - 1, stolb = indexStart.Y + 1; str >= end.X; str--, stolb++)

if (str == end.X && stolb == end.Y) { onDiagonal = true; break; }

//на диагонали

if (onDiagonal)

{//дойти до панели диагональю

for (int str = indexStart.X - 1, stolb = indexStart.Y + 1; str > end.X; str--, stolb++)

{//искать другие фигуры на пути

if (figure[str, stolb] != null) return "Путь перекрыт";

} return "OK";

} return "help";

}

//ниже //панель левее?

if (indexStart.Y > end.Y)

{//идти диагонально по 1 клетке //подтвердить что панель лежит на диагонали

for (int str = indexStart.X + 1, stolb = indexStart.Y - 1; str <= end.X; str++, stolb--)

if (str == end.X && stolb == end.Y) { onDiagonal = true; break; }

//на диагонали

if (onDiagonal)

{//дойти до панели диагональю

for (int str = indexStart.X + 1, stolb = indexStart.Y - 1; str < end.X; str++, stolb--)

{//искать другие фигуры на пути

if (figure[str, stolb] != null) return "Путь перекрыт";

} return "OK";

} return "help";

}

//правее//идти диагонально по 1 клетке //подтвердить что панель лежит на диагонали

for (int str = indexStart.X + 1, stolb = indexStart.Y + 1; str <= end.X; str++, stolb++)

if (str == end.X && stolb == end.Y) { onDiagonal = true; break; }

//на диагонали

if (onDiagonal)

{//дойти до панели диагональю

for (int str = indexStart.X + 1, stolb = indexStart.Y + 1; str < end.X; str++, stolb++)

{//искать другие фигуры на пути

if (figure[str, stolb] != null) return "Путь перекрыт";

} return "OK";

} return "help";

}

/// <summary>

/// Для коня методика хода.

/// Конечная панель выбрана верно?

/// Т.к. фигура перескакивает через другие.

/// </summary>

private string GoKoni(Point indexStart, Point end, bool black)

{

//панель выше?

if(indexStart.X > end.X)

{//панель левее\правее на один фигуры?

if (indexStart.Y - 1 == end.Y || indexStart.Y + 1 == end.Y)

if (indexStart.X - 2 == end.X) return "OK";

//панель левее\правее на два фигуры?

if (indexStart.Y - 2 == end.Y || indexStart.Y + 2 == end.Y)

if (indexStart.X - 1 == end.X) return "OK";

}

else//панель ниже

{//панель левее\правее на один фигуры?

if (indexStart.Y - 1 == end.Y || indexStart.Y + 1 == end.Y)

if (indexStart.X + 2 == end.X) return "OK";

//панель левее\правее на два фигуры?

if (indexStart.Y - 2 == end.Y || indexStart.Y + 2 == end.Y)

if (indexStart.X + 1 == end.X) return "OK";

}

return "help";

}

/// <summary>

/// Для ладьи методика хода.

/// Могу ли ладьей дойти до панели end?

/// И нет ли фигур на всем пути исключая саму панель.

/// </summary>

private string GoLaden(Point indexStart, Point end, bool black)

{

//определить уровень

if(indexStart.X == end.X)

{//лево право

//перейти надо в лево?

if(indexStart.Y > end.Y)

{//идти от фигуры до панели по 1 клетке

for (int i = indexStart.Y-1; i > end.Y; i--)

{//если клетка не пуста, то

if (figure[indexStart.X, i] != null) return "Путь перекрыт";

} return "OK";

}

//в право

for (int i = indexStart.Y + 1; i < end.Y; i++)

{//если клетка не пуста, то

if (figure[indexStart.X, i] != null) return "Путь перекрыт";

} return "OK";

}

if(indexStart.Y == end.Y)

{//верх низ

//перейти надо в верх?

if (indexStart.X > end.X)

{//идти от фигуры до панели по 1 клетке

for (int i = indexStart.X - 1; i > end.X; i--)

{//если клетка не пуста, то

if (figure[i, indexStart.Y] != null) return "Путь перекрыт";

}return "OK";

}

//в низ

for (int i = indexStart.X + 1; i < end.X; i++)

{//если клетка не пуста, то

if (figure[i, indexStart.Y] != null) return "Путь перекрыт";

} return "OK";

}

return "help";

}

/// <summary>

/// Для пешек методика хода.

/// Могу ли пешкой дойти до панели end.

/// И нет ли фигур на всем пути исключая саму панель.

/// </summary>

private string GoPawn(Point indexStart, Point end, bool black, bool pawnBatle)

{

//если pawnBatle == true то анализируем диагонали лево 1 И право 1

if(!pawnBatle)

{//если pawnBatle == false то анализируем верх 1 И низ 1

if (indexStart.Y == end.Y)//в одну линию

{

if (black && indexStart.X > end.X)

{

//черные и фигура ниже панели

if ((indexStart.X - end.X) <= 1)

/\*От фигуры до панели одна клетка

\*То переходим к шагу просмотра

\*-->"Есть ли обьект на панели?" \*/

return "OK";

if ((indexStart.X - end.X) == 2 && !figure[indexStart.X, indexStart.Y].OneGo)

/\*От фигуры до панели 2 клетки и фигура без запрета хода на две клетки

\*Проверить клетку перед фигурой и до панели:\*/

if (figure[indexStart.X - 1, indexStart.Y] == null)

//клетка пуста

return "OK";

//то переходим к шагу просмотра

/\*-->"Есть ли обьект на панели?"\*/

else return "Путь перекрыт";

//от фигуры до панели долековато:

return "help";

}

if (!black && indexStart.X < end.X)

{

//белые и фигура ниже панели

if ((end.X - indexStart.X) <= 1)

return "OK";

if ((end.X - indexStart.X) == 2 && !figure[indexStart.X, indexStart.Y].OneGo)

if (figure[indexStart.X + 1, indexStart.Y] == null)

return "OK";

else return "Путь перекрыт";

return "help";

}

}else return "help";

}

else

{//диагонали:

if(black)

{//верхние направление

//панель выше фигуры

if(end.X < indexStart.X)

{

if (indexStart.X - 1 == end.X && indexStart.Y - 1 == end.Y) return "OK";

if (indexStart.X - 1 == end.X && indexStart.Y + 1 == end.Y) return "OK";

return "help";

}

return "err";

}

if(!black)

{//нижнее направление

//панель ниже фигуры

if (end.X > indexStart.X)

{

if (indexStart.X + 1 == end.X && indexStart.Y - 1 == end.Y) return "OK";

if (indexStart.X + 1 == end.X && indexStart.Y + 1 == end.Y) return "OK";

return "help";

}

return "err";

}

}

return "help";

}

/// <summary>

/// Глянуть по индексам в массиве поле OneGo фигуры

/// </summary>

public bool OneGoFigure(Point start)

{

return figure[start.X, start.Y].OneGo;

}

/// <summary>

/// Скопировать массив в лист

/// </summary>

internal IEnumerable<string> CopyFiguresDataToList()

{

List<string> list = new List<string>();

foreach (clFigure clFigure in figure)

{

if (clFigure != null)

list.Add(clFigure.Name + " " + clFigure.OneGo);

else list.Add("null");

}

return list;

}

/// <summary>

/// Переписать лист в массив

/// </summary>

internal void PastDataListToFiguresArr(IEnumerable<string> dataList)

{

int i = 0, j = -1;

foreach (string str in dataList)

{

j++;

if(j < 8)

{

if (str == "null")

{

figure[i, j] = null;

}

else

{

string[] words = str.Split(' ');

if (figure[i, j] == null) figure[i, j] = new clFigure(words[0]);

else figure[i, j] = new clFigure(words[0]);

//figure[i, j].Name = words[0];

//поставить поле два шага для пешек

figure[i, j].OneGo = !str.Contains("False");

}

}

else

{

j = 0; i++;

if (str == "null")

{

figure[i, j] = null;

}

else

{

string[] words = str.Split(' ');

if (figure[i, j] == null) figure[i, j] = new clFigure(words[0]);

else figure[i, j] = new clFigure(words[0]);

//поставить поле два шага для пешек

figure[i, j].OneGo = !str.Contains("False");

}

}

}

}

/// <summary>

/// Переписать лист в массив

/// </summary>

internal void PastDataListToFiguresArr(IEnumerable<clFigure> dataList)

{

int i = 0, j = -1;

foreach (var listFig in dataList)

{

j++;

if (j < 8)

{

if (listFig == null)

{

figure[i, j] = null;

}

else

{

if (figure[i, j] == null) figure[i, j] = new clFigure(listFig.Name);

else

{

figure[i, j] = listFig;

//поставить поле два шага для пешек

figure[i, j].OneGo = listFig.OneGo;

}

}

}

else

{

j = 0; i++;

if (listFig == null)

{

figure[i, j] = null;

}

else

{

if (figure[i, j] == null) figure[i, j] = new clFigure(listFig.Name);

else

{

figure[i, j] = listFig;

//поставить поле два шага для пешек

figure[i, j].OneGo = listFig.OneGo;

}

}

}

}

}

/// <summary>

/// Вернуть текущее состояние массива

/// </summary>

internal IEnumerable<clFigure> FigureObjArrState()

{

var fig = new List<clFigure>();

foreach (var clFigure in figure)

fig.Add(clFigure);

return fig;

}

}

}

* 1. **Класс clFigure – описание фигуры как объекта:**

namespace Шахматы

{

class clFigure

{

private readonly string \_nameFigure;

/// <summary>

/// Имя обьекта

/// </summary>

public string Name { get; set; }

/// <summary>

/// Поле для пешки - первый ход разрешено на две клетки.

/// False - доступно два хода

/// </summary>

public bool OneGo { get; set; }

/// <summary>

/// Когда пешка превращается в фигуру высокого ранга

/// </summary>

public string TransformFigurePawn { get; set; }

/// <summary>

/// создать персональную фигуру

/// </summary>

public clFigure(string name)

{

\_nameFigure = name;

Name = \_nameFigure;

//если обьект - пешка, то понадобится поле ПЕРВЫЙ\_ХОД

if(name.Contains("ПЕШКА")) Pawn();

}

/// <summary>

/// Первый ход на 1//2 клетки

/// </summary>

private void Pawn()

{//ход разрешен

OneGo = false;

}

/\*

Кнопку на форме с записью положения в массиве связываем

\* при помощи поля Tag кнопки.

\* У каждой панели поле Tag даёт адрес в массиве\на форме

\* типо .Tag=40 означает ячейку в массиве[4,0]

\*/

}

}

* 1. **Класс clXod – отвечает за логику последовательности хода:**

namespace Шахматы

{

class clXod

{

/// <summary>

/// Отметка о выборе фигуры для хода

/// </summary>

public bool XodStart { get; set; }

/// <summary>

/// Отметка о завершении хода

/// </summary>

public bool XodEnd { get; set; }

/// <summary>

/// Тег фигуры, которой совершено начало хода

/// </summary>

public string XodStartFigureTag { get; set; }

/// <summary>

/// Очередность ходов

/// </summary>

public bool BlackGo { get; set; }

/// <summary>

/// Первыми ходят ЧЕРНЫЕ?

/// </summary>

public clXod(bool goBlack)

{//true первый ход - черных\\иначе - белых

BlackGo = goBlack;

}

/// <summary>

/// Сбросить ход.

/// </summary>

public void ResetXodState()

{

if(XodStart && XodEnd)

{

XodStart = false;

XodEnd = false;

XodStartFigureTag = "";

}

}

/// <summary>

/// Выдернуть часть названия фигуры

/// </summary>

public string ConteinsFigure(string xodStartFigureTag)

{

//определить - чё за кнопка?//тип фигуры

if (xodStartFigureTag.Contains("ПЕШКА")) return "ПЕШКА";

if (xodStartFigureTag.Contains("ЛАДЬЯ")) return "ЛАДЬЯ";

if (xodStartFigureTag.Contains("КОНЬ")) return "КОНЬ";

if (xodStartFigureTag.Contains("СЛОН")) return "СЛОН";

if (xodStartFigureTag.Contains("КАРОЛЬ")) return "КАРОЛЬ";

if (xodStartFigureTag.Contains("ФЕРЗЬ")) return "ФЕРЗЬ";

return "null";

}

}

}

* 1. **Класс fMenu – отвечает за настройки игры:**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

using DevComponents.DotNetBar;

using DevComponents.DotNetBar.Controls;

using Шахматы.Properties;

namespace Шахматы

{

public partial class fMenu

{

public fMenu()

{

InitializeComponent();

OneGo = radioBtn\_GoBlack.Checked ? true : false;

\_help = true;

PodsvetOn = radioButton1.Checked ? true : false;

ShaxOn = rBtn\_OnShax.Checked ? true : false;

fAbout();

BoardImage = null;

reflectionImage5.BackgroundImage = Properties.Resources.fig\_deff\_mini;

reflectionImage5.SizeMode = PictureBoxSizeMode.CenterImage;

lblInfoStyle.Text = Resources.fMenu\_SelectedTypeFigures\_Click\_QwinCor\_Style\_Pak;

StateSelectImgPak = 0;

}

void fAbout(){textBox1.Select(0, 0);}

/// <summary>

/// Кто первый пойдет в игре

/// </summary>

public bool OneGo { get; private set; }

/// <summary>

/// Ключи на запись:

/// </summary>

private bool \_keyUpB, \_keyDownB, \_keyLeftB, \_keyRightB,

\_keyUpW, \_keyDownW, \_keyLeftW, \_keyRightW;

/// <summary>

/// public string[] \_black = new string[3], \_white = new string[3];

/// </summary>

public string[,] \_arr = new string[2,4];

/// <summary>

/// Нажат один из четырех лейблов Черные назначить кнопку

/// </summary>

private void blackKeyChenged\_Click(object sender, EventArgs e)

{

var lblObj = (sender as LabelX);

//нажали - хочим сменить

//изменить центральную рактинку типо ЗАПИСЬ

lblStartWritingKeyBlack.BackgroundImage = global::Шахматы.Properties.Resources.achievement10\_rollover;

//установить флаг на разрешение отлова нажатой клавиши:

if (lblObj != null)

switch (lblObj.Name)

{

case "lblBlackUp":

{

\_keyUpB = true;

}

break;

case "lblBlackDown":

{

\_keyDownB = true;//\_keyDownB = true;

}

break;

case "lblBlackLeft": \_keyLeftB = true;

break;

case "lblBlackRight": \_keyRightB = true;

break;

}

}

/// <summary>

/// Нажат один из четырех лейблов Белые назначить кнопку

/// </summary>

private void whiteKeyChenged\_Click(object sender, EventArgs e)

{

lblStartWritingKeyWhite.BackgroundImage = global::Шахматы.Properties.Resources.achievement10\_rollover;

var lblObj = (sender as LabelX);

//нажали - хочим сменить

//установить флаг на разрешение отлова нажатой клавиши:

if (lblObj != null)

switch (lblObj.Name)

{

case "lblWhiteUp":

{

\_keyUpW = true;

}

break;

case "lblWhiteDown":

{

\_keyDownW = true;//\_keyDownB = true;

}

break;

case "lblWhiteLeft": \_keyLeftW = true;

break;

case "lblWhiteRight": \_keyRightW = true;

break;

}

}

/// <summary>

/// Само назначение клавиш, покачто их надо кудато запоминать

/// </summary>

private void fMenu\_KeyUp(object sender, KeyEventArgs e)

{

var key = e.KeyValue;

var tip = e.KeyCode;

tabControlPanel3.Focus();

if(\_keyUpB)

{

lblStartWritingKeyBlack.BackgroundImage = global::Шахматы.Properties.Resources.achievement10;

toolTipForKeys.SetToolTip(this.lblBlackUp, tip.ToString());

\_keyUpB = !\_keyUpB;

\_arr[0, 0] = key.ToString();

return;

}

if (\_keyDownB)

{

lblStartWritingKeyBlack.BackgroundImage = global::Шахматы.Properties.Resources.achievement10;

toolTipForKeys.SetToolTip(this.lblBlackDown, tip.ToString());

\_keyDownB = !\_keyDownB;

\_arr[0, 1] = key.ToString();

return;

}

if (\_keyLeftB)

{

lblStartWritingKeyBlack.BackgroundImage = global::Шахматы.Properties.Resources.achievement10;

toolTipForKeys.SetToolTip(this.lblBlackLeft, tip.ToString());

\_keyLeftB = !\_keyLeftB;

\_arr[0, 2] = key.ToString();

return;

}

if (\_keyRightB)

{

lblStartWritingKeyBlack.BackgroundImage = global::Шахматы.Properties.Resources.achievement10;

toolTipForKeys.SetToolTip(this.lblBlackRight, tip.ToString());

\_keyRightB = !\_keyRightB;

\_arr[0, 3] = key.ToString();

return;

}

#region white if

if (\_keyUpW)

{

lblStartWritingKeyWhite.BackgroundImage = global::Шахматы.Properties.Resources.achievement10;

toolTipForKeys.SetToolTip(this.lblWhiteUp, tip.ToString());

\_keyUpW = !\_keyUpW;

\_arr[1, 0] = key.ToString();

return;

}

if (\_keyDownW)

{

lblStartWritingKeyWhite.BackgroundImage = global::Шахматы.Properties.Resources.achievement10;

toolTipForKeys.SetToolTip(this.lblWhiteDown, tip.ToString());

\_keyDownW = !\_keyDownW;

\_arr[1, 1] = key.ToString();

return;

}

if (\_keyLeftW)

{

lblStartWritingKeyWhite.BackgroundImage = global::Шахматы.Properties.Resources.achievement10;

toolTipForKeys.SetToolTip(this.lblWhiteLeft, tip.ToString());

\_keyLeftW = !\_keyLeftW;

\_arr[1, 2] = key.ToString();

return;

}

if (\_keyRightW)

{

lblStartWritingKeyWhite.BackgroundImage = global::Шахматы.Properties.Resources.achievement10;

toolTipForKeys.SetToolTip(this.lblWhiteRight, tip.ToString());

\_keyRightW = !\_keyRightW;

\_arr[1, 3] = key.ToString();

return;

}

#endregion

}

/// <summary>

/// Вернет true при назначении новых кнопок

/// </summary>

public bool IsNewKeys { get; set; }

/// <summary>

/// Если сменилось изображение, то было переназначение кнопок управления

/// </summary>

private void lblStartWritingKeyBlack\_BackgroundImageChanged(object sender, EventArgs e)

{

IsNewKeys = true;

}

/// <summary>

/// Изменен порядок ходов Ч\Б

/// </summary>

private void radioBtn\_GoBlack\_CheckedChanged(object sender, EventArgs e)

{

OneGo = radioBtn\_GoBlack.Checked;

\_state = true;

}

private bool \_state;

/// <summary>

/// True если изменилось значение выбора очередности хода

/// </summary>

public bool IsEvent() {return \_state;}

private bool \_help;

/// <summary>

/// True Включено отображение подсказок

/// </summary>

public bool IsShowHelp() {return \_help;}

private void rBtn\_OnHelp\_CheckedChanged(object sender, EventArgs e)

{

\_help = rBtn\_OnHelp.Checked;

}

/// <summary>

/// Обновить данные на форме статистики

/// </summary>

internal void Statistic(int count)

{

lbl\_vsegoIgr.Text = count.ToString();

}

/// <summary>

/// Обновить состояние графиков статистики Ч\Б

/// </summary>

internal void UpdateLiveChart(int countBlackWin)

{

microChart1.DataPoints = new List<double>(new double[] { 1, 32, 30, 20, 55, 66 });

microChart2.DataPoints = new List<double>(new double[] { 99, 68, 70, 80, 45, 34 });

int count = 0;//колич игр

if (!string.IsNullOrEmpty(lbl\_vsegoIgr.Text))

count = Convert.ToInt32(lbl\_vsegoIgr.Text);

//колич побед Черных

try

{

int winB = (countBlackWin \* 100) / count;

int winW = ((count - countBlackWin) \* 100) / count;

microChart1.DataPoints.Add(winB);

microChart2.DataPoints.Add(winW);

}

catch (DivideByZeroException)

{

int winB = (countBlackWin \* 100) / 1;

int winW = ((count - countBlackWin) \* 100) / 1;

microChart1.DataPoints.Add(winB);

microChart2.DataPoints.Add(winW);

}

microChart1.InvalidateChart(true);

microChart2.InvalidateChart(true);

}

/// <summary>

/// True включает подсветку хода

/// </summary>

public bool PodsvetOn { get; private set; }

/// <summary>

/// Подсвечивать ходы?

/// </summary>

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

PodsvetOn = radioButton2.Checked;

PodsvetOn = radioButton1.Checked;

}

/// <summary>

/// True если включено отслеживание ШАХа

/// </summary>

public bool ShaxOn {get; private set; }

/// <summary>

/// Отслеживать угрозу королю в виде ШАХа?

/// </summary>

private void rBtn\_OnShax\_CheckedChanged(object sender, EventArgs e)

{

ShaxOn = rBtn\_OffShax.Checked;

ShaxOn = rBtn\_OnShax.Checked;

}

private clImagePakFigures \_imgNumTypePak = new clImagePakFigures(0);

/// <summary>

/// Клик по набору фигурок, смена изображение в reflectionImage5

/// Выставляем набор фигурок.

/// </summary>

private void SelectedTypeFigures\_Click(object sender, EventArgs e)

{

var btnItm = sender as ButtonItem;

if(btnItm!=null)

switch (btnItm.Tag.ToString())

{

case "0"://fig\_deff.png

\_imgNumTypePak = new clImagePakFigures(0);

FigImgPak = new Dictionary<string, Image>(\_imgNumTypePak.ImgPak);

reflectionImage5.BackgroundImage = Properties.Resources.fig\_deff\_mini;

lblInfoStyle.Text = Resources.fMenu\_SelectedTypeFigures\_Click\_QwinCor\_Style\_Pak;

StateSelectImgPak = 0;

break;

case "1"://fig1.png

\_imgNumTypePak = new clImagePakFigures(1);

FigImgPak = new Dictionary<string, Image>(\_imgNumTypePak.ImgPak);

reflectionImage5.BackgroundImage = Properties.Resources.fig1;

lblInfoStyle.Text = Resources.fMenu\_SelectedTypeFigures\_Click\_Korolev\_Style\_Pak;

StateSelectImgPak = 2;

break;

case "2"://fig\_mini.png

\_imgNumTypePak = new clImagePakFigures(2);

FigImgPak = new Dictionary<string, Image>(\_imgNumTypePak.ImgPak);

reflectionImage5.BackgroundImage = Properties.Resources.fig\_mini;

lblInfoStyle.Text = Resources.fMenu\_SelectedTypeFigures\_Click\_Korolev\_Style\_Pak;

StateSelectImgPak = 1;

break;

}

else

{

var tmp = sender.ToString();

\_imgNumTypePak = new clImagePakFigures(0);

FigImgPak = new Dictionary<string, Image>(\_imgNumTypePak.ImgPak);

StateSelectImgPak = 0;

}

}

/// <summary>

/// тег бел\_пешка\_а7 переделать в бел\_пешка+индекс

/// найти и вернуть

/// </summary>

internal Image SearchAndTransform(string btnTeg, int index)

{

if(FigImgPak == null)

SelectedTypeFigures\_Click("--", null);

string tmp;

if(index > 0) tmp = (btnTeg.Remove(btnTeg.LastIndexOf('\_'))) + index;

else tmp = (btnTeg.Remove(btnTeg.LastIndexOf('\_')));

if (!FigImgPak.ContainsKey(tmp.ToLower()))

{

string[] arr = btnTeg.Split('\_');

tmp = arr[0] + '\_' + \_imgNumTypePak.TransformToAssociativFig(arr[1]) + index;

}

return (from image in FigImgPak where image.Key == tmp.ToLower() select image.Value).FirstOrDefault();

}

/// <summary>

/// Передать пакет фигурок

/// </summary>

private Dictionary<string, Image> FigImgPak { get; set; }

/// <summary>

/// Один из набора radionButton в положении true

/// Выставляем доску

/// </summary>

private void rbSelectedBoadrType\_Click(object sender, EventArgs e)

{

var checkRbtn = (sender as RadioButton);

if(checkRbtn!=null)

switch (checkRbtn.Tag.ToString())

{

case "0": BoardImage = Properties.Resources.board\_deff;

break;

case "1": BoardImage = Properties.Resources.board;

break;

case "2": BoardImage = Properties.Resources.board1;

break;

case "3": BoardImage = Properties.Resources.board2;

break;

}

else

{

var checkImage = (sender as ReflectionImage);

if (checkImage != null)

switch (checkImage.Tag.ToString())

{

case "0": BoardImage = Properties.Resources.board\_deff;

rb\_BoardStyleDeff.Checked = true;

break;

case "1": BoardImage = Properties.Resources.board;

rb\_BoardStyle\_0.Checked = true;

break;

case "2": BoardImage = Properties.Resources.board1;

rb\_BoardStyle\_1.Checked = true;

break;

case "3": BoardImage = Properties.Resources.board2;

rb\_BoardStyle\_2.Checked = true;

break;

}

}

}

/// <summary>

/// Передаем выставленное изображение доски

/// </summary>

internal Image BoardImage { get; private set; }

/// <summary>

/// Дабы не напрягать проц, делаем флаг на запрос "нужна смена пакета"

/// </summary>

internal int StateSelectImgPak { get; private set; }

}

}

* 1. **Класс fTransformPawnToN – отвечает за диалог выбора фигуры:**

using DevComponents.DotNetBar;

namespace Шахматы

{

/// <summary>

/// Пешка превращается в другую ранговую фигуру

/// </summary>

public partial class fTransformPawnToN

{

public fTransformPawnToN()

{

InitializeComponent();

Increment = 0; Increment++;

OpisanieTXT("btnILaden");

}

/// <summary>

/// Общая форма "btnInn\_n"

/// </summary>

private int Increment;

/// <summary>

/// Принудительно повысить инкремент новой фигуры

/// </summary>

internal void UpIncrement()

{

Increment++;

SelectedFigure = \_figName + "\_" + Increment;

}

private void OpisanieTXT(string figType)

{

switch (figType)

{

case "btnILaden":

opisaniePlus.Text = "ход на дальняк";

opisanieMinus.Text = "ход только по прямой";

break;

case "btnIKoni":

opisaniePlus.Text = "взятие радиальных барьеров";

opisanieMinus.Text = "важна позиция";

break;

case "btnISlon":

opisaniePlus.Text = "ход на дальняк";

opisanieMinus.Text = "ход только по диагонали\n\rход по клеткам цвета фигуры";

break;

case "btnIFerz":

opisaniePlus.Text = "безбашенная фигура";

opisanieMinus.Text = "нет";

break;

}

}

/// <summary>

/// Ознакомление с ттх фигуры

/// </summary>

private void btnILaden\_Click(object sender, System.EventArgs e)

{

opisanieMinus.Text = " \n\r ";

var figure = sender as ButtonItem;

if(figure!=null)

OpisanieTXT(figure.Name);

}

/// <summary>

/// Подтверждение выбранной фигуры

/// </summary>

private void btnIFerz\_DoubleClick(object sender, System.EventArgs e)

{

var figure = sender as ButtonItem;

if (figure != null)

{

SelectedFigure = figure.Name+"\_"+Increment;

\_figName = figure.Name;

this.Close();

}

}

private string \_figName;

/// <summary>

/// Выбранная фигура

/// </summary>

public string SelectedFigure { get; private set; }

}

}

* 1. **Класс clImagePakFigures –ресурсы для визуализации фигур:**

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using Шахматы.Properties;

namespace Шахматы

{

/// <summary>

/// Так как есть несколько стилей фигур-изображений, то

/// необходим регулятор, который укажет необходимый набор.

/// </summary>

class clImagePakFigures

{

#region Firures

private readonly Dictionary<string, Image> FigPakDeff = new Dictionary<string, Image>

{

{"бел\_конь", Resources.белый\_конь},

{"бел\_кароль", Resources.белый\_король},

{"бел\_ладья", Resources.белый\_ладья},

{"бел\_пешка", Resources.белый\_пешка},

{"бел\_слон", Resources.белый\_слон},

{"бел\_ферзь", Resources.белый\_ферзь},

{"чер\_конь", Resources.черный\_конь},

{"чер\_кароль", Resources.черный\_король},

{"чер\_ладья", Resources.черный\_ладья},

{"чер\_пешка", Resources.черный\_пешка},

{"чер\_слон", Resources.черный\_слон},

{"чер\_ферзь", Resources.черный\_ферзь}

};

private readonly Dictionary<string, Image> FigPakClassik = new Dictionary<string, Image>

{

{"бел\_конь1", Resources.белый\_конь1},

{"бел\_кароль1", Resources.белый\_король1},

{"бел\_ладья1", Resources.белый\_ладья1},

{"бел\_пешка1", Resources.белый\_пешка1},

{"бел\_слон1", Resources.белый\_слон1},

{"бел\_ферзь1", Resources.белый\_ферзь1},

{"чер\_конь1", Resources.черный\_конь1},

{"чер\_кароль1", Resources.черный\_король1},

{"чер\_ладья1", Resources.черный\_ладья1},

{"чер\_пешка1", Resources.черный\_пешка1},

{"чер\_слон1", Resources.черный\_слон1},

{"чер\_ферзь1", Resources.черный\_ферзь1}

};

private readonly Dictionary<string, Image> FigPakAutor = new Dictionary<string, Image>

{

{"бел\_конь2", Resources.белый\_конь2},

{"бел\_кароль2", Resources.белый\_король2},

{"бел\_ладья2", Resources.белый\_ладья2},

{"бел\_пешка2", Resources.белый\_пешка2},

{"бел\_слон2", Resources.белый\_слон2},

{"бел\_ферзь2", Resources.белый\_ферзь2},

{"чер\_конь2", Resources.черный\_конь2},

{"чер\_кароль2", Resources.черный\_король2},

{"чер\_ладья2", Resources.черный\_ладья2},

{"чер\_пешка2", Resources.черный\_пешка2},

{"чер\_слон2", Resources.черный\_слон2},

{"чер\_ферзь2", Resources.черный\_ферзь2}

};

private readonly Dictionary<string, string> FigAssociative = new Dictionary<string, string>

{

{"btnIKoni", "конь" },

{"btnILaden", "ладья" },

{"btnISlon", "слон" },

{"btnIFerz", "ферзь" }

};

#endregion

public clImagePakFigures(int typePak)

{

CountFigArrPak = FigPakDeff.Count;

switch (typePak)

{

case 0: ImgPak = new Dictionary<string, Image>(FigPakDeff);

break;

case 1: ImgPak = new Dictionary<string, Image>(FigPakAutor);

break;

case 2: ImgPak = new Dictionary<string, Image>(FigPakClassik);

break;

}

}

/// <summary>

/// На вход тег типо "btnIFerz" на выходе "ферзь"

/// </summary>

internal string TransformToAssociativFig(string associative)

{

return (from variable in FigAssociative where variable.Key == associative select variable.Value).FirstOrDefault();

}

internal int CountFigArrPak { get; private set; }

internal readonly Dictionary<string, Image> ImgPak;

}

}

* 1. **Класс KeyControlsDeff – отвечает за управление с клавиатуры:**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using System.Xml;

namespace Шахматы

{

/// <summary>

/// Либо закреплять и каждый раз управление ставить через меню

///либо хранить умолчания\новые клавиши в файле

///и при старте проги инициализировать этот класс чтобы

///считать управление а при переназначении в меню - перепиать

///новые клавиши вместо старых в файл.XML?

/// </summary>

public class KeyControlsDeff

{

/// <summary>

/// Конструктор для Чтения/доступа

/// </summary>

public KeyControlsDeff()

{

ReadFromFile();

}

//MessageBox.Show(roses[index].Name);//Показать Name Tag xml

private List<Rose> roses = new List<Rose>();//масив роз + явный шаблон класса Rose

private List<Rose> lilian = new List<Rose>();

/// <summary>

/// Получить значение для [цвет фигуры] по индексу [0..3]

/// return KeyValuePair (Name--Value)

/// </summary>

public KeyValuePair<string, string> ReadParam(string colorFig, int index)

{

if (index >= 0 && index <= 3)

return (colorFig == "black")

? new KeyValuePair<string, string>(roses[index].Name, roses[index].Value)

: new KeyValuePair<string, string>(lilian[index].Name, lilian[index].Value);

//roses[index].Value : lilian[index].Value;

return new KeyValuePair<string, string>("null", "null"); ;

}

/// <summary>

/// Записать новые кнопы заместо старых в XML

/// </summary>

/// <param name="sqrtTag">black или white</param>

/// <param name="index">0..3</param>

/// <param name="value">new key</param>

public void WriteParam(string sqrtTag, int index, string value)

{

bool key = false;

//изменить содержимое массива

if (index >= 0 && index <= 3)

if (sqrtTag == "black")

{//Rose

roses[index].Value = value; key = true;

}

else if (sqrtTag == "white")

{//White

lilian[index].Value = value; key = true;

}

//записать изменения в файл

if (key) SaveToFile();

}

/// <summary>

/// Записать новые кнопы заместо старых в XML

/// </summary>

public void WriteParam(string[,] arr)

{//int index, string value

//изменить содержимое массива

for (int i = 0; i < 4; i++)

{

if (arr[0, i]!=null)

roses[i].Value = arr[0, i];

}

for (int i = 0; i < 4; i++)

{

if (arr[1, i]!=null)

lilian[i].Value = arr[1, i];

}

//записать изменения в файл

if (true) SaveToFile();

}

/// <summary>

/// Считать комбо из файла keyControl.xml

/// </summary>

private void ReadFromFile()

{

FileStream fs = null;

try { fs = new FileStream("keyControl.xml", FileMode.OpenOrCreate); }

catch { MessageBox.Show("Нет файла разметки keyControl.xml"); }

XmlTextReader xmlIn = new XmlTextReader(fs);

xmlIn.WhitespaceHandling = WhitespaceHandling.None;//Откл обработку пробелов

xmlIn.MoveToContent();//Переместиться в начало дока

if (xmlIn.Name != "black")//Проверим первый тег дока

throw new ArgumentException("Тег открытия xml повреждён или изменён.");

//цикл чтения документа <Black></Black>

do

{

if (!xmlIn.Read())

throw new ArgumentException("Ошибка чтения xml структуры файла.");

if ((xmlIn.NodeType == XmlNodeType.EndElement) & (xmlIn.Name == "black"))

break;//Это конец Дока, элемент black И завершающий\_элемент

if (xmlIn.NodeType == XmlNodeType.EndElement)

continue;//Если эт <конечный /> элемент, то не проверяем

switch (xmlIn.Name)

{

case "Rose":

{//если это роза, то читаем ее параметры

Rose newItem = new Rose("", 0, "");

roses.Add(newItem);

newItem.LoadFromFile(xmlIn);

}

break;

case "White":

{//если это роза, то читаем ее параметры

Rose newItem = new Rose("", 0, "");

lilian.Add(newItem);

newItem.LoadFromFile(xmlIn);

}

break;

}

} while (!xmlIn.EOF);

//закрываем потоки

xmlIn.Close();

if (fs != null) fs.Close();

}

/// <summary>

/// Сохранить\переписать кнопки в файл xml

/// </summary>

void SaveToFile()

{

//создание потока записи и обьекта создания xml-документа

FileStream fs = null;

try { fs = new FileStream("keyControl.xml", FileMode.OpenOrCreate); }

catch { MessageBox.Show("Нет файла разметки keyControl.xml"); }

if (fs != null)

{

XmlTextWriter xmlOut = new XmlTextWriter(fs, Encoding.Unicode);

xmlOut.Formatting = Formatting.Indented;

//старт начала документа

xmlOut.WriteStartDocument();

xmlOut.WriteComment("Этот файл - собственность QwinCor");

xmlOut.WriteComment("qwinmen@yandex.ru");

//создаем корневой элемент

xmlOut.WriteStartElement("black");

xmlOut.WriteAttributeString("Version", DateTime.Now.ToShortDateString());

//перебор всех роз и сохранение их

foreach (Rose item in roses)

item.SaveToFile(xmlOut, "Rose");

foreach (Rose item in lilian)

item.SaveToFile(xmlOut, "White");

//закрываем корневой тег и документ

xmlOut.WriteEndElement();

xmlOut.WriteEndDocument();

//закрываем обьекты записи

xmlOut.Close();

fs.Close();

}

}

/// <summary>

/// Вернет "индекс" если в направлении Key можно поднятся

/// Иначе null

/// </summary>

public string Go(string key, string bevelStateIndex)

{

const int up = 0, down = 7, left = 0, right = 7;

if (bevelStateIndex == null) return null;

int one = Convert.ToInt32(bevelStateIndex[0].ToString());

if (key == "up")

{//идти вверх, тоесть на строку ниже 44--34

//берем положение квадрата и переписываем в новое

one -= 1;

if (one < up) return null;

return (one + "" + bevelStateIndex[1]);

}

if (key == "down")

{

//берем положение квадрата и переписываем в новое

one += 1;

if (one > down) return null;

return (one + "" + bevelStateIndex[1]);

}

one = Convert.ToInt32(bevelStateIndex[1].ToString());

if (key == "left")

{

//берем положение квадрата и переписываем в новое

one -= 1;

if (one < left) return null;

return (bevelStateIndex[0] + "" + one);

}

if (key == "right")

{

//берем положение квадрата и переписываем в новое

one += 1;

if (one > right) return null;

return (bevelStateIndex[0] + "" + one);

}

return null;

}

}

}

* 1. **Класс Rose – отвечает за работу с файлом настроек xml:**

using System;

using System.Xml;

namespace Шахматы

{

class Rose

{

/// <summary>

/// Храним полученные keysControl из xml

/// </summary>

public Rose(string name, int index, string value)

{

this.Name = name;

this.Index = index;

this.Value = value;

}

public string Name { get; set; }

public int Index { get; set; }

public string Value { get; set; }

public void LoadFromFile(XmlTextReader xmlIn)

{

try

{

Name = xmlIn.GetAttribute("Name");

Index = Convert.ToInt32(xmlIn.GetAttribute("Index"));

Value = xmlIn.GetAttribute("Value");

}

catch { throw new ArgumentException("Ошибка сохранения атрибутов при чтении xml"); }

}

public void SaveToFile(XmlTextWriter xmlOut, string startAtrib)

{

try

{

xmlOut.WriteStartElement(startAtrib);

xmlOut.WriteAttributeString("Name", Name);

xmlOut.WriteAttributeString("Index", Index.ToString());

xmlOut.WriteAttributeString("Value", Value.ToString());

xmlOut.WriteEndElement();

}

catch (Exception)

{

throw new Exception("Запись в файл прервана");

}

}

}

}

* 1. **Класс clStack – отвечает за логику совершения откатов в игре:**

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

namespace Шахматы

{

/// <summary>

/// Обеспечить механизм создания\заполнения\перемещения\удаления

/// контейнера для среза "массив фигур"

/// </summary>

class clStack

{

/// <summary>

/// Снимок обьекта "массив"

/// </summary>

private List<List<clFigure>> \_stackArr;

/// <summary>

/// Хей ход был когда делался снимок в стек

/// </summary>

private List<bool> \_state;

internal int IndexPos { get; set; }

private string[,] \_list;

/// <summary>

/// Какой индекс у последнего элемента

/// </summary>

internal int CountIndex { get; private set; }

/// <summary>

/// Обьект типа clFigure[,]

/// </summary>

public clStack()

{

\_stackArr = new List<List<clFigure>>(20);

\_state = new List<bool>(20);

\_list = new string[21,64];

CountIndex = 0;

}

internal List<clFigure> this[int index]

{

get { return GetIndexObject(index); }

set { \_stackArr[index] = value; }

}

/// <summary>

/// Добавить обьект в конец списка

/// </summary>

internal void Add(List<clFigure> objFigure, bool xod)

{

if (\_stackArr.Count < 20)

{

\_stackArr.Add(objFigure);

\_state.Add(xod);

CountIndex++;

}

else

{

CountIndex = 0;

DeliteOne();

Add(objFigure, xod);//\_stackArr.Add(keyValuePair);

}

}

/// <summary>

/// Обрезать массив

/// </summary>

private void DeliteOne()

{

//вырезать с 10 по 20 элементы и поместить их с 0 по 10

var tmpList = \_stackArr.GetRange(10, 10);//10-10

\_stackArr.Clear();

\_stackArr = tmpList;

//вырезать с 10 по 20 элементы и поместить их с 0 по 10

var tmp = \_state.GetRange(10, 10);//10-10

\_state.Clear();

\_state = tmp;

}

private int \_i = 0;

/// <summary>

/// Добавить List\_clFigure\_ в конец списка

/// </summary>

internal void AddString(IEnumerable<clFigure> objFigure, bool xod)

{

int j = 0;

if(\_i < 21)

{

\_state.Add(xod);

foreach (clFigure figure in objFigure)

{

if (figure == null) \_list[\_i, j] = "null";

else \_list[\_i, j] = figure.Name + " " + figure.OneGo;

j++;

}

\_i++; CountIndex++;

}

else

{

\_i = 10;

CountIndex = 0;

DeliteOneString();

AddString(objFigure, xod);//\_stackArr.Add(keyValuePair);

}

}

/// <summary>

/// Добавить List\_string\_ в конец списка

/// </summary>

internal void AddString(IEnumerable<string> objFigure, bool xod)

{

if(objFigure==null) return;

int j = 0;

if (\_i < 21)

{

\_state.Add(xod);

foreach (string figure in objFigure)

{

if (figure == "null") \_list[\_i, j] = "null";

else \_list[\_i, j] = figure;

j++;

}

\_i++; CountIndex++;

}

else

{

\_i = 10;

CountIndex = 0;

DeliteOneString();

AddString(objFigure, xod);//\_stackArr.Add(keyValuePair);

}

}

/// <summary>

/// Элементы с 10 по 20 врезать на место 0..10

/// Остальное с 11 по 20 занулить

/// </summary>

private void DeliteOneString()

{

int s = 10, l = 0;

while (l < 11)

{

for (int t = 0; t < 64; t++)

{//[0..10, 0..64] = [10..20, 0..64]

\_list[l, t] = \_list[s, t];

}

l++; s++;

}

for (int j = 10; j < 21; j++)

{

for (int i = 0; i < 64; i++)

{//[10..20, 0..64] = null

\_list[j, i] = null;

}

}

}

/// <summary>

/// Добавить обьект в N позицию списка

/// </summary>

internal void AddToN(List<clFigure> keyValuePair, int index)

{

this[index] = keyValuePair;

}

/// <summary>

/// Получить начальный элемент списка

/// </summary>

internal List<clFigure> GetFirstObject()

{

return \_stackArr.FirstOrDefault();

}

/// <summary>

/// Получить последний элемент списка

/// </summary>

internal List<clFigure> GetLastObject()

{

return \_stackArr.LastOrDefault();

}

/// <summary>

/// Получить последний элемент-состояние-хода списка

/// </summary>

internal bool GetLastBoolObject()

{

return \_state.LastOrDefault();

}

/// <summary>

/// Получить указанный элемент списка

/// </summary>

internal List<clFigure> GetIndexObject(int index)

{

return \_stackArr.ElementAtOrDefault(index);

}

/// <summary>

/// Получить указанный элемент-состояние-хода списка

/// </summary>

internal bool GetBoolObject(int index)

{

return \_state.ElementAtOrDefault(index);

}

/// <summary>

/// Получить обьект по индексу

/// </summary>

internal IEnumerable<string> GetStringListObject(int index)

{

if (index < 0) return null;

List<string> tmp = new List<string>();

for (int i = 0; i < 64; i++)

{

tmp.Add(\_list[index, i]);

}

if (tmp.TakeWhile(s => s == null).Any())

{

return null;

}

return tmp;

}

/// <summary>

/// Получить последний элемент списка

/// </summary>

internal IEnumerable<string> GetLastListObject()

{

var tmp = new List<string>();

for (int i = 0; i < 64; i++)

{

tmp.Add(\_list[this.CountIndex-1, i]);

}

return tmp;

}

}

}

* 1. **Класс clWorkFile – отвечает за сохранение\загрузку игры:**

using System;

using System.Collections.Generic;

using System.IO;

namespace Шахматы

{

class clWorkFile

{

public clWorkFile(){ }

/// <summary>

/// Количество запусков за сеанс

/// </summary>

public int Schetchik { get; set; }

/// <summary>

/// Количество выйгрышей черных

/// </summary>

public int BlackWin { get; set; }

/// <summary>

/// Сохранить в файл

/// </summary>

public void Save()

{

StreamWriter sw = File.CreateText("save.txt");

//в начало пишем колич игр

sw.WriteLine(Schetchik);

//пишем результат игры

sw.WriteLine(BlackWin);

sw.Close();

}

private List<string > \_arr = new List<string>();

/// <summary>

/// Подгрузить файл

/// </summary>

public void Load()

{

\_arr.Clear();

StreamReader sr = new StreamReader("save.txt");

while (true)

{

string str = sr.ReadLine();

if(str==null)

break;

\_arr.Add(str);

}

sr.Close();

InShetchikLoad();

InBlackWin();

}

/// <summary>

/// Перегнать строку в целое и поместить в счетчик загрузок

/// </summary>

private void InShetchikLoad()

{

if (\_arr != null)

Schetchik = Convert.ToInt32(\_arr[0]);

}

/// <summary>

/// Перегнать строку в целое и поместить в счетчик побед

/// </summary>

private void InBlackWin()

{

if (\_arr != null)

BlackWin = Convert.ToInt32(\_arr[1]);

}

}

}

* 1. **Класс clCopyAsValue – переводит ссылки в значения:**

using System.Collections.Generic;

using System.Windows.Forms;

namespace Шахматы

{

/// <summary>

/// При обычном копировании обьектов типо "кнопка--лист--лист" мы работаем с сылками на корень.

/// Нам же нужно работать с независимыми значениями обьекта в его первом виде записи в лист.

/// </summary>

class clCopyAsValue

{

public readonly List<KeyValuePair<string, string>> ObjValue;

public clCopyAsValue()

{

ObjValue = new List<KeyValuePair<string, string>>();

}

/// <summary>

/// Добавить в лист обьект по значению

/// </summary>

internal void AddList(ButtonBase objLink)

{

if (objLink == null) return;

ObjValue.Add(new KeyValuePair<string, string>(objLink.Name, objLink.Tag.ToString()));

}

}

}