**Лабораторная работа 4. 2 часть**

**Игровое приложение «Пятнашки».**

Новое GUI.

В данной лабораторной работе нужно изменить интерфейс приложения Пятнашки, вернее создать еще три дополнительных интерфейса.

Первый интерфейс уже был создан в первой части лабораторной работы. Это стандартный статический WindowsFormsApplication.

Для начала отделим логику игры от описания интерфейса. Создайте в решении новый проект GameLibrary (Библиотека классов C#) и вынесите в него все классы, связанные с игрой, в том числе классы для отмены хода.

Чтобы предыдущий проект работал, добавьте в его раздел Ссылки ссылку на проект GameLibrary (рис. 1)

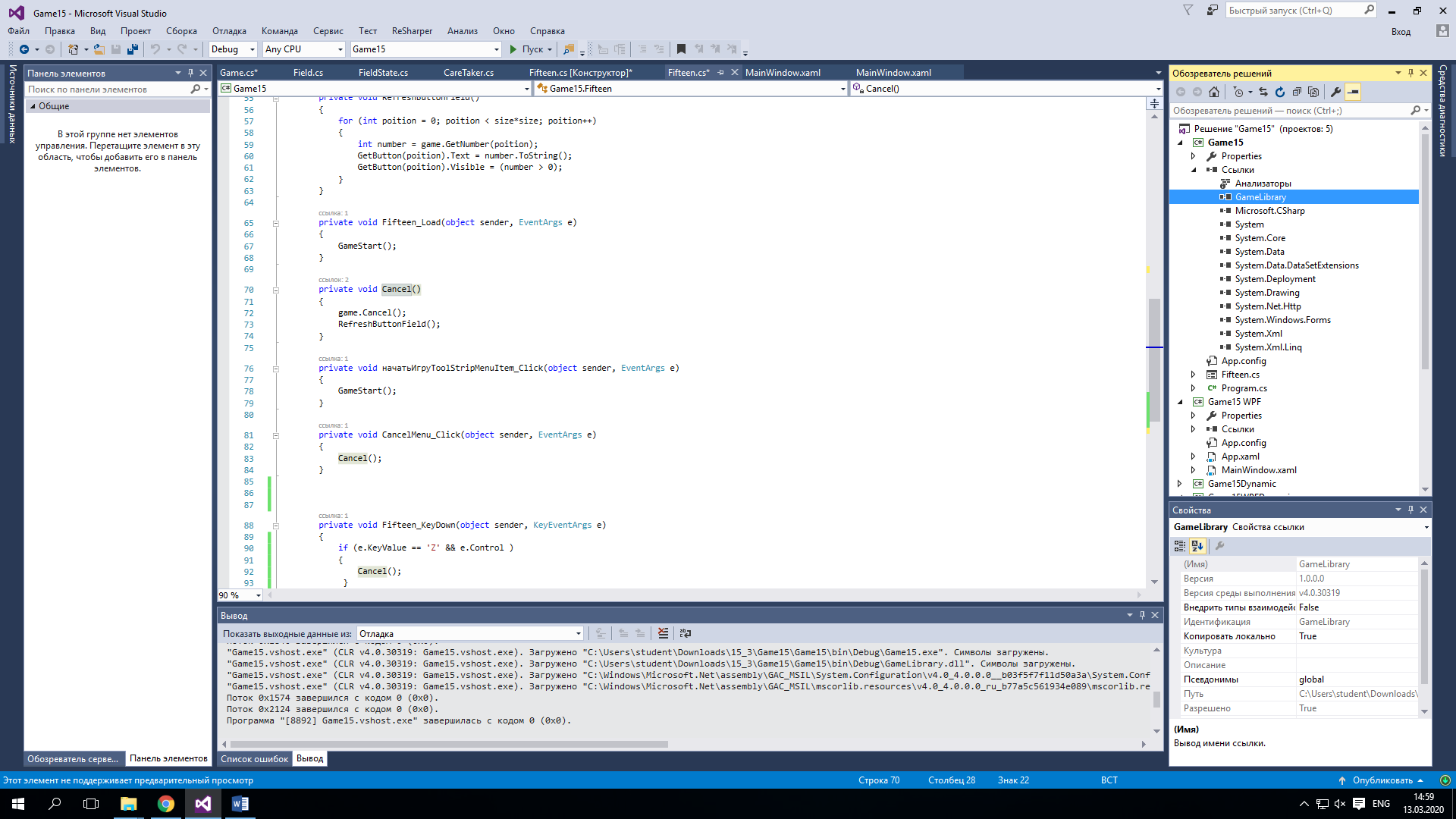


Рисунок 1. Ссылки проекта

**Запуск**. Запустите проект и убедитесь, что он остался рабочим.

Добавьте к решению новый проект Приложение WPF с именем FifteenWPF. Откройте файл разметки – файл с расширением xaml.

По умолчанию файл разметки имеет следующее содержимое:

<Window x:Class=" FifteenWPF.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace: FifteenWPF"

mc:Ignorable="d"

Title="MainWindow" Height="450" Width="500">

<Grid>

</Grid>

</Window>

Здесь есть основное окно и контейнер компоновки – **Grid**.

Задайте форме следующие свойства

|  |  |
| --- | --- |
| **Свойство** | **Значение** |
| Title | Пятнашки |
| Height | 500 |
| Width | 460 |

Рассмотрим один из вариантов построения графического интерфейса. Замените контейнер Grid контейнером DockPanel. Этот контейнер пристыковывает элементы к одной из своих границ. Важен порядок следования дочерних элементов.

Установите свойству LastChildFill контейнера DockPanel значение true. Это будет означать, что последний элемент данного контейнера заполнит все оставшееся пространство.

<DockPanel LastChildFill="True">

</DockPanel>

Внутри данного контейнера расположите компоненты Menu и UniformGrid. Каждому дочернему (вложенному) элементу нужно задать способ расположения. Свойство DockPanel.Dock указывает, к какому краю контейнера должен пристыковаться данный элемент управления. Возможные значения данного свойства: Left, Right, Top, Bottom.

Установите данное свойство компонентам Menu и UniformGrid.

Внутри Menu создайте элемент:

<MenuItem Name="MenuStartGame" Header="Начать игру" FontSize="16" Click="StartGame">

</MenuItem>

Свойство Name позволит обратиться к этому элементу в программе, свойство Header озаглавит данный пункт меню. Кроме того, можно задать размер шрифта и добавить обработчик события на нажатие (выбор) данного пункта меню. После того, как будет записан обработчик, среда разработки сама предложит создать Новый обработчик события (рис. 2)

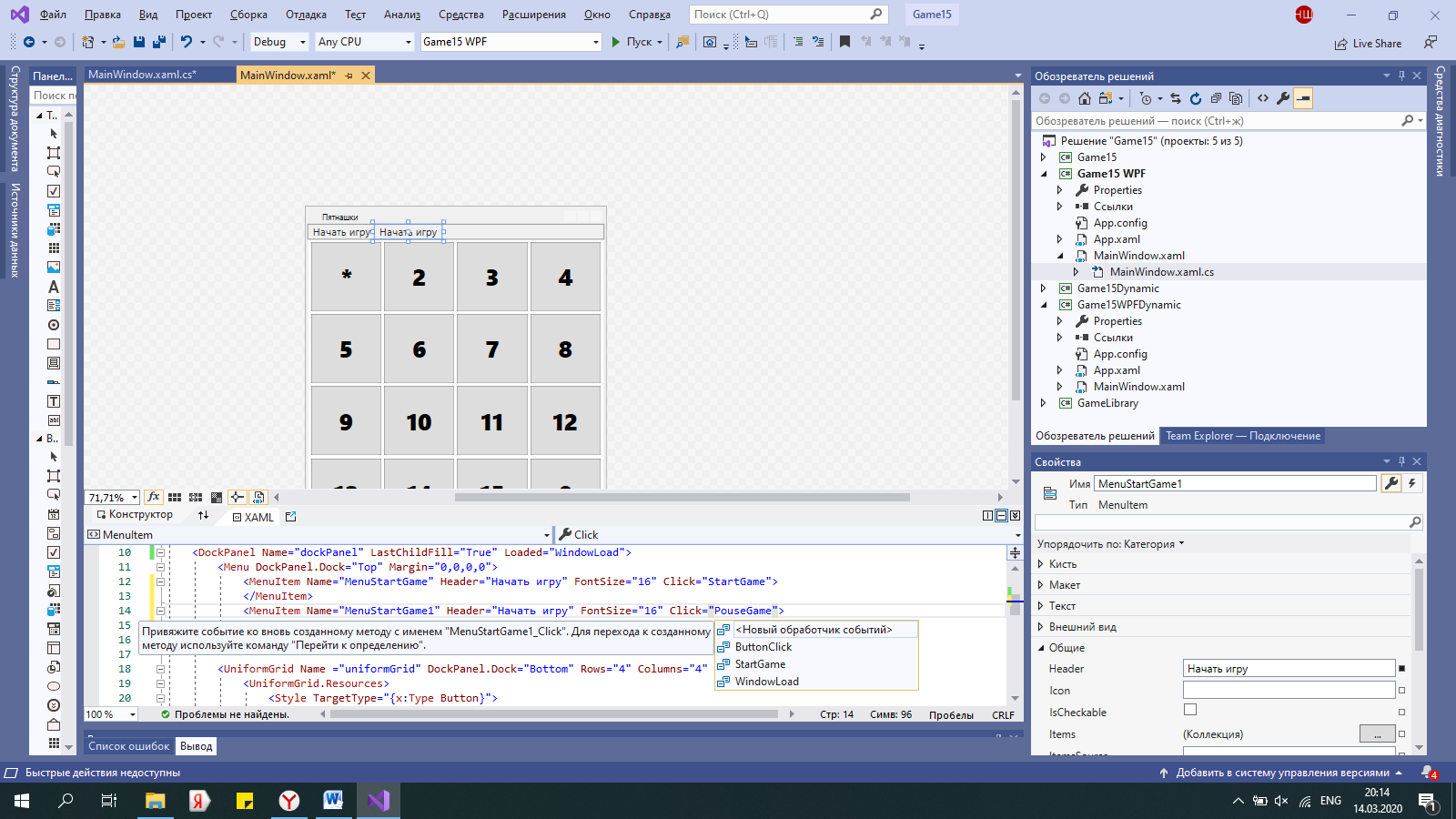


Рисунок 2. Контекстное меню создания элемента

В классе MainWindow будет создан StartGame:

private void StartGame(object sender, RoutedEventArgs e)

{

}

Описание класса MainWindow находится в файле MainWindow.xaml.cs. Его можно открыть, нажав F7 на файле MainWindow.xaml в обозревателе решений.

Сетка UniformGrid отличается от обычной сетки Grid тем, что все ячейки в ней имеют одинаковый размер. Задайте следующие свойства:

|  |  |
| --- | --- |
| **Свойство** | **Значение** |
| Name | uniformGrid |
| DockPanel.Dock | Bottom |
| Rows | 4 |
| Columns | 4 |
| Margin | 2,2,2,2 |

Для размещения элементов внутри этого контейнера не нужно задавать им расположение (строку и колонку). Они помещаются друг за другом в порядке их добавления в контейнер.

Добавьте 16 кнопок. Каждая кнопка должна иметь уникальное имя и свойство Tag:

<Button Name ="button0" Tag="0"/>

Каждой кнопке нужно задать ряд одинаковых свойств – общий стиль. Все общие свойства удобно собрать в ресурсах:

<UniformGrid.Resources>

<Style TargetType="{x:Type Button}">

<Setter Property="Focusable" Value="False"/>

<EventSetter Event="Click" Handler="ButtonClick"/>

</Style>

</UniformGrid.Resources>

Добавьте к списку одинаковых свойств:

* **Content** , например, пустую строку;
* **FontSize**;
* **Margin**;
* выберите подходящее значение для свойства **FontWeight** и т.п.

Настройте внешний вид по своему усмотрению, ориентируясь на пример (рис. 3).

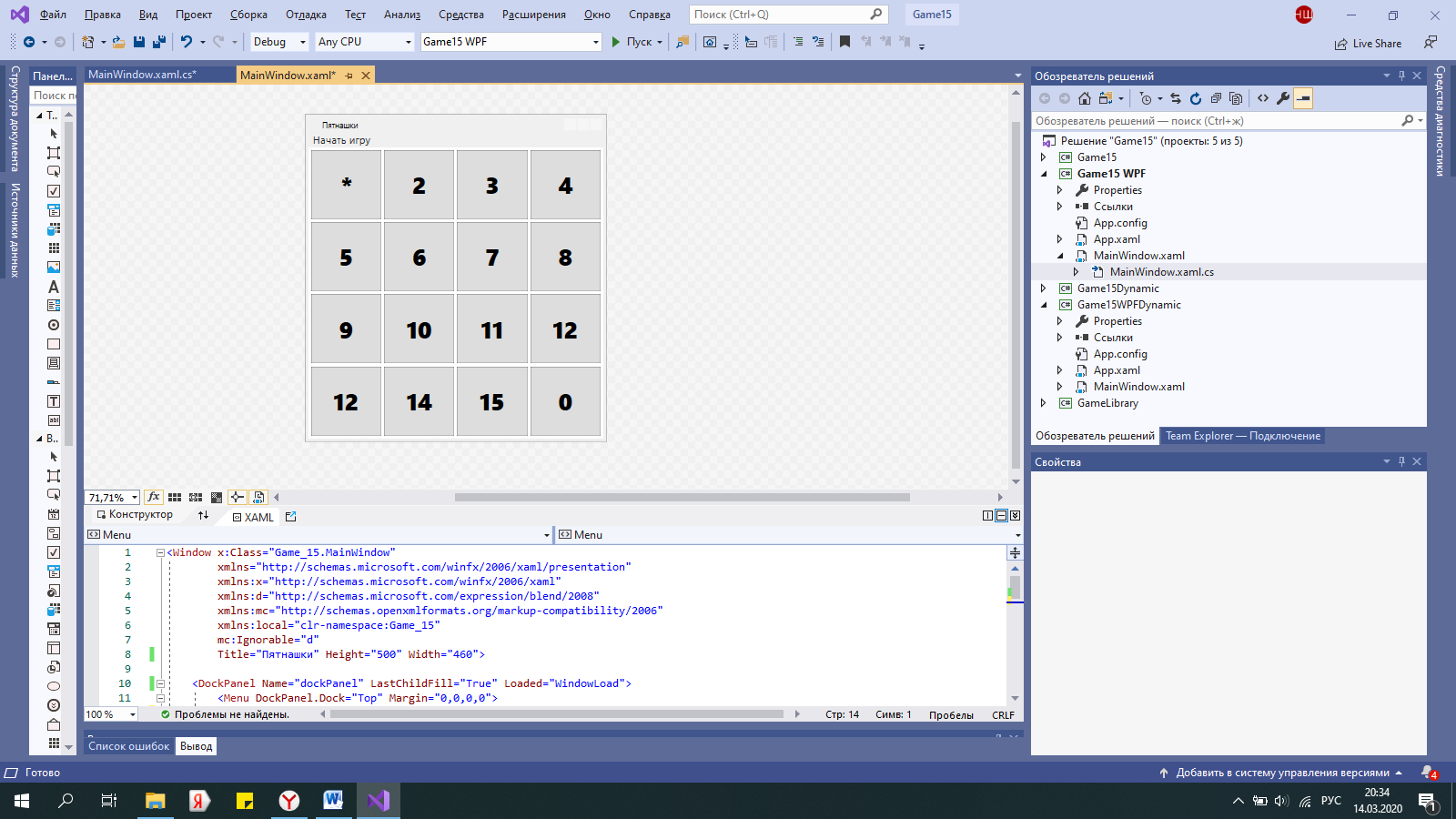


Рисунок 3. Окно для игры Пятнашки

Опишите обработчик события ButtonClick для кнопки. Дополните класс MainWindow методами, как в классе Fifteen.

**Запуск**. Запустите проект и убедитесь в его работоспособности.

Оба интерфейса ограничены в количестве созданных кнопок. Для того, чтобы исправить этот недостаток, нужно создавать интерфейс динамически, в классе.

Сейчас решение имеет три проекта, добавьте четвертый проект Windows Forms.

Чтобы макет был «резиновым» используйте контейнер TableLayoutPanel:

TableLayoutPanel maintlp = new TableLayoutPanel()

{

Dock = DockStyle.Fill,

ColumnCount = 1,

RowCount = 2,

};

Добавьте две строки и установите для них стиль. Примеры стилей:

maintlp.RowStyles.Add(new RowStyle(SizeType.Absolute, число));

maintlp.RowStyles.Add(new RowStyle(SizeType.AutoSize));

maintlp.RowStyles.Add(new RowStyle(SizeType.Percent, 100 / size));

Справочный материал: <https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.forms.tablelayoutpanel?view=netframework-4.8>.

В верхней строке расположите MenuStrip, в нижней – еще один элемент TableLayoutPanel для поля игры. Количество строк и столбцов можно задавать в программе.

Задание.

1. Доработайте интерфейс приложения. Добавьте возможность изменять размер поля.
2. Добавьте к решению еще одно Приложение WPF для динамического интерфейса игры Пятнашки. Справочный материал: <https://metanit.com/sharp/wpf/4.2.php>, <https://docs.microsoft.com/ru-ru/dotnet/api/system.windows.controls.grid?view=netframework-4.8>.

**Замечание**. В динамических интерфейсах изменяемой частью является только поле. Поэтому можно создавать в программе только поле.