

Создание программного приложения на языке C# (WPF)

Цель: ознакомиться с процессом разработки программного обеспечения в среде Visual Studio, на языке C# с использованием технологии WPF, на примере разработки интерактивного теста.

Программное обеспечение: среду разработки Visual Studio Community можно бесплатно получить по ссылке <https://visualstudio.microsoft.com/ru/vs/community/> (**примечание:** если вы выполняете данную работу в учебной аудитории, Visual Studio уже должен быть установлен на ваших ПК). При установке достаточно выбрать “Разработка классических приложений .NET”.

Часть 0: описание проекта

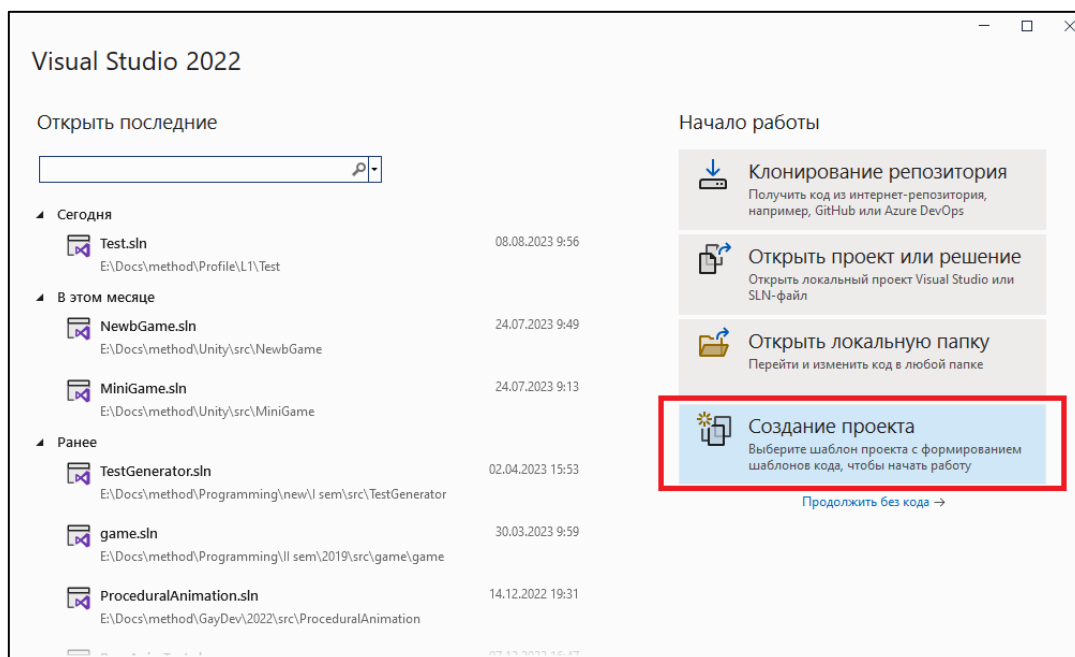
Целью работы является разработка игрового приложения, создающего тесты на скорость и внимательность. Внешний вид приложения представлен на изображении ниже:



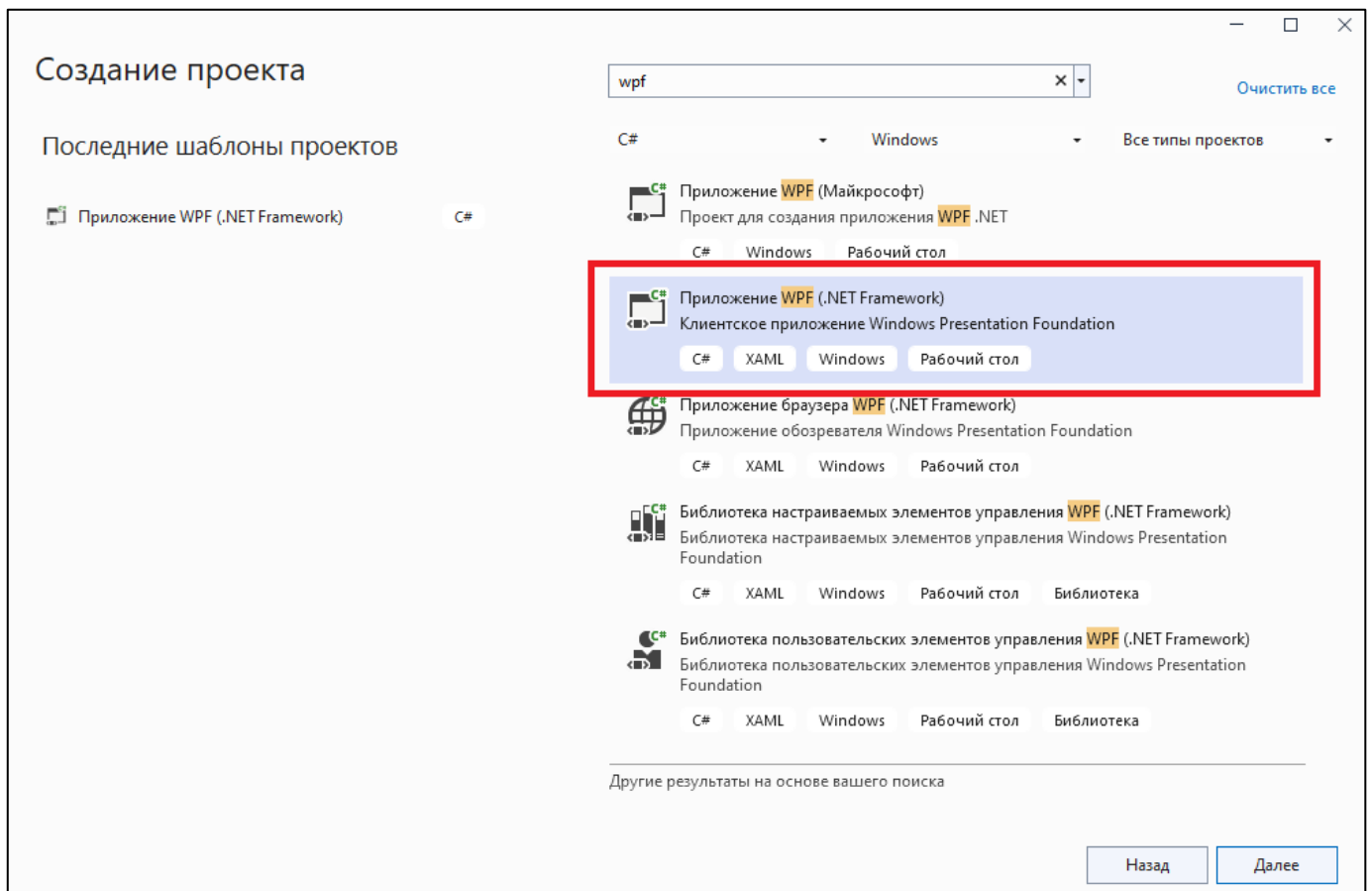
При нажатии кнопки “Start”, генерируется игровое поле и запускается таймер. Игровое поле состоит из кнопок, пронумерованных по возрастанию. Игроку необходимо как можно быстрее последовательно нажать кнопки в порядке возрастания их номеров. При нажатии кнопок в правильной последовательности, число открытых полей увеличивается, при ошибке, число открытых полей не изменяется, а к прошедшему времени добавляется одна секунда.

Часть 1: создание проекта

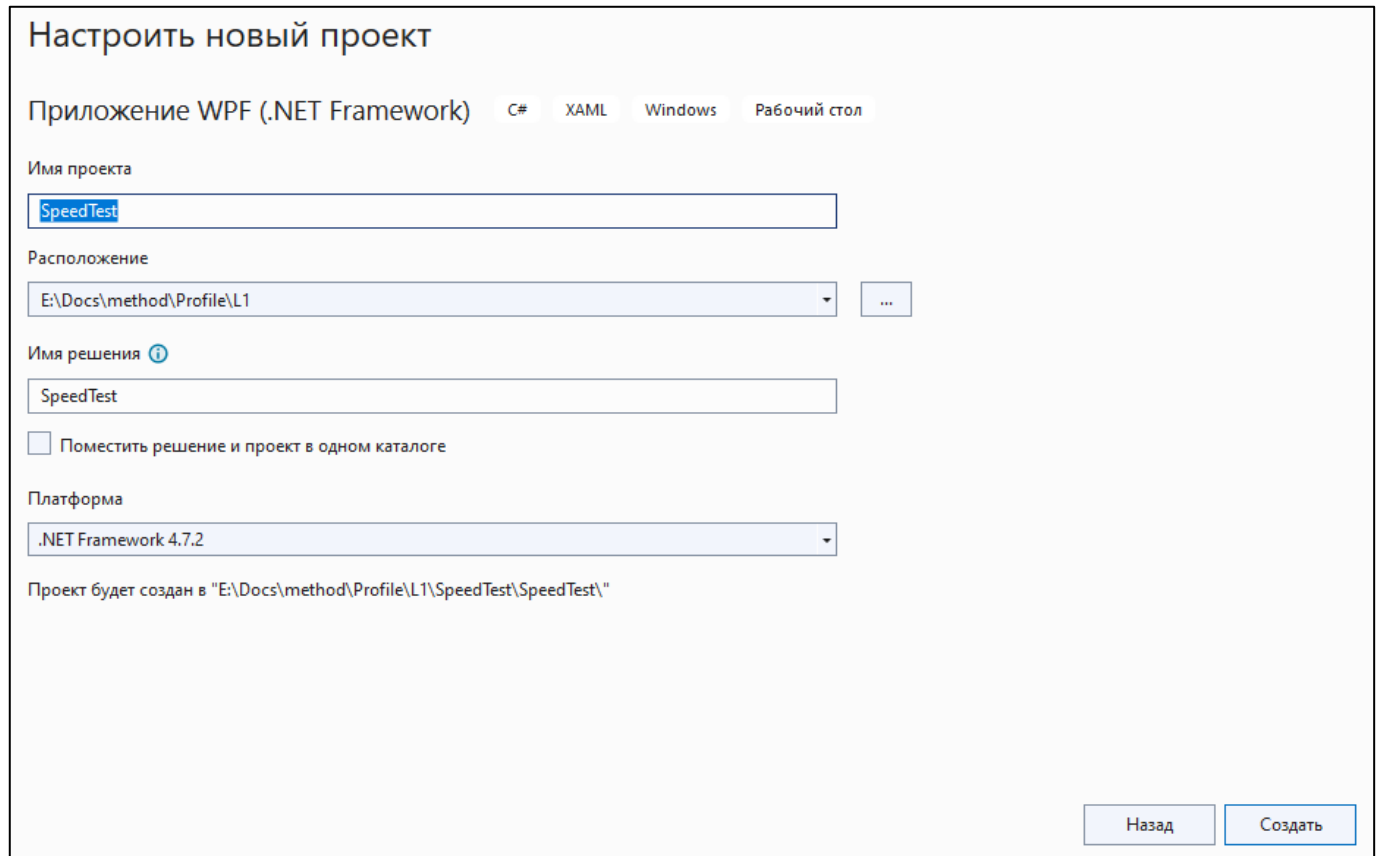
Запустите Visual Studio и кликните “Создание проекта”:



Выберите “Приложение WPF (.NET Framework)” и нажмите “Далее”:

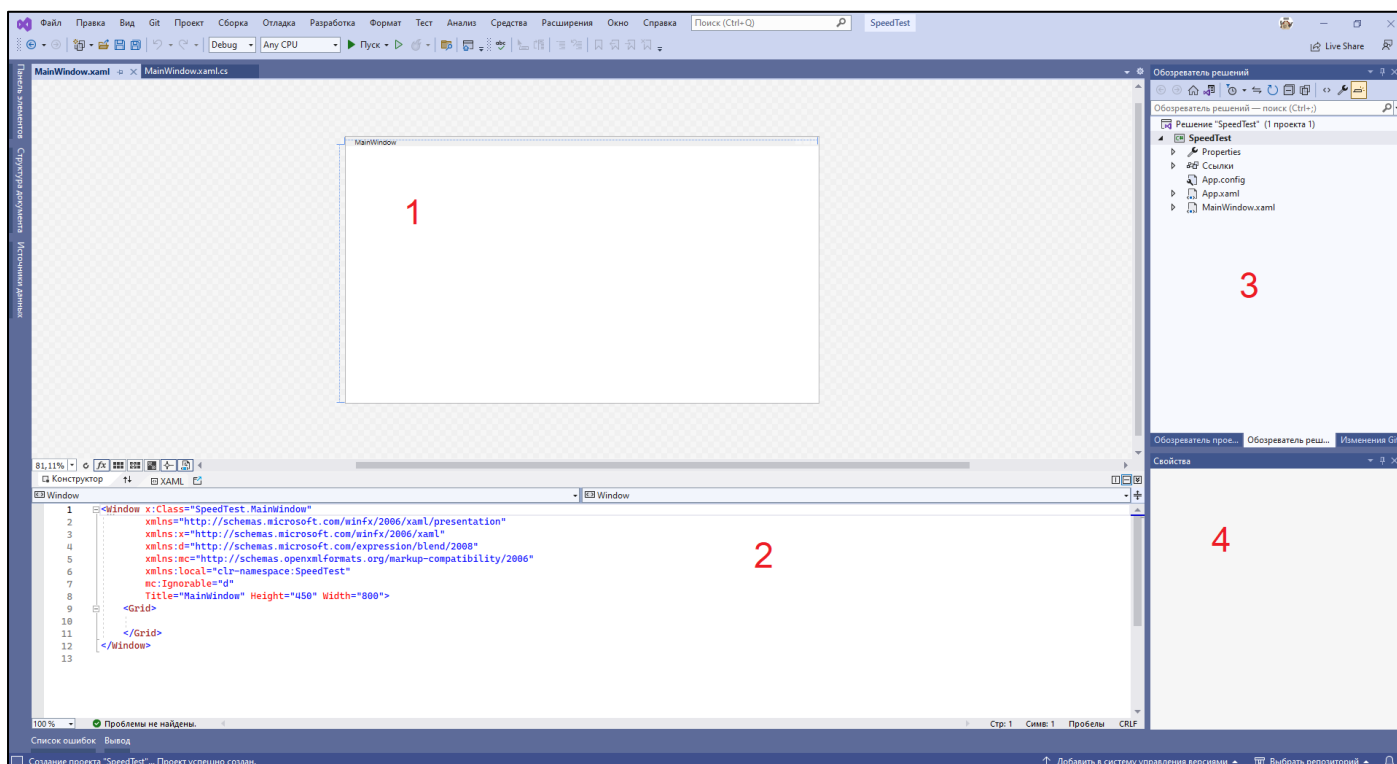


В появившемся окне, введите название проекта, выберите расположение и нажмите “Создать”:



Примечание: не рекомендуется создавать проекты на внешних носителях (USB), это значительно влияет на время отклика при работе с проектом.

После создания проекта, откроется окно, которое можно, условно, поделить на 4 части:



Примечание: если в вашем проекте отсутствует окно свойств, или обозреватель решений, включить их можно в выпадающем меню “Вид”.

1 – визуальный редактор окна (формы) вашего приложения. На этой форме будут размещаться различные элементы управления и отображаться информация.

2 – XAML редактор. Описание вашей формы на специализированном языке разметки. Редактировать вашу форму можно как через визуальный редактор, так и при помощи текста.

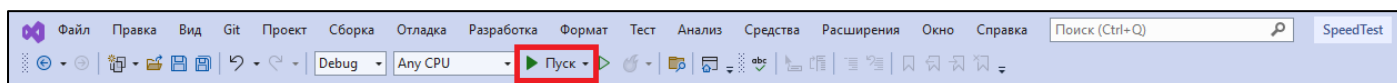
3 – обозреватель решений. В этом разделе показывается структура и файлы вашего проекта.

4 – окно свойств. В этом окне показаны свойства и события элемента формы, выбранного в текущий момент.

Используя окно свойств, или XAML редактор, можно сразу же сменить заголовок главного окна вашего приложения:

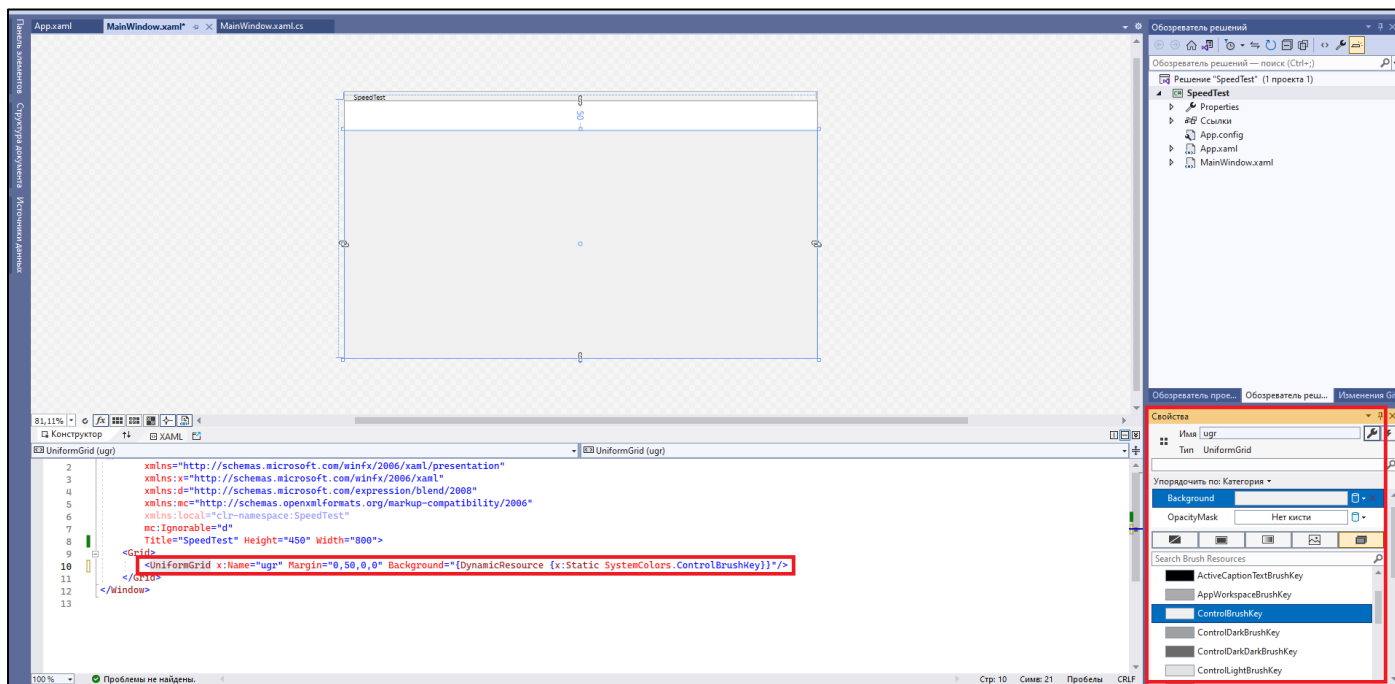


Запустить приложение можно нажав кнопку “Пуск”, клавишу F5, или воспользовавшись меню “Отладка”:



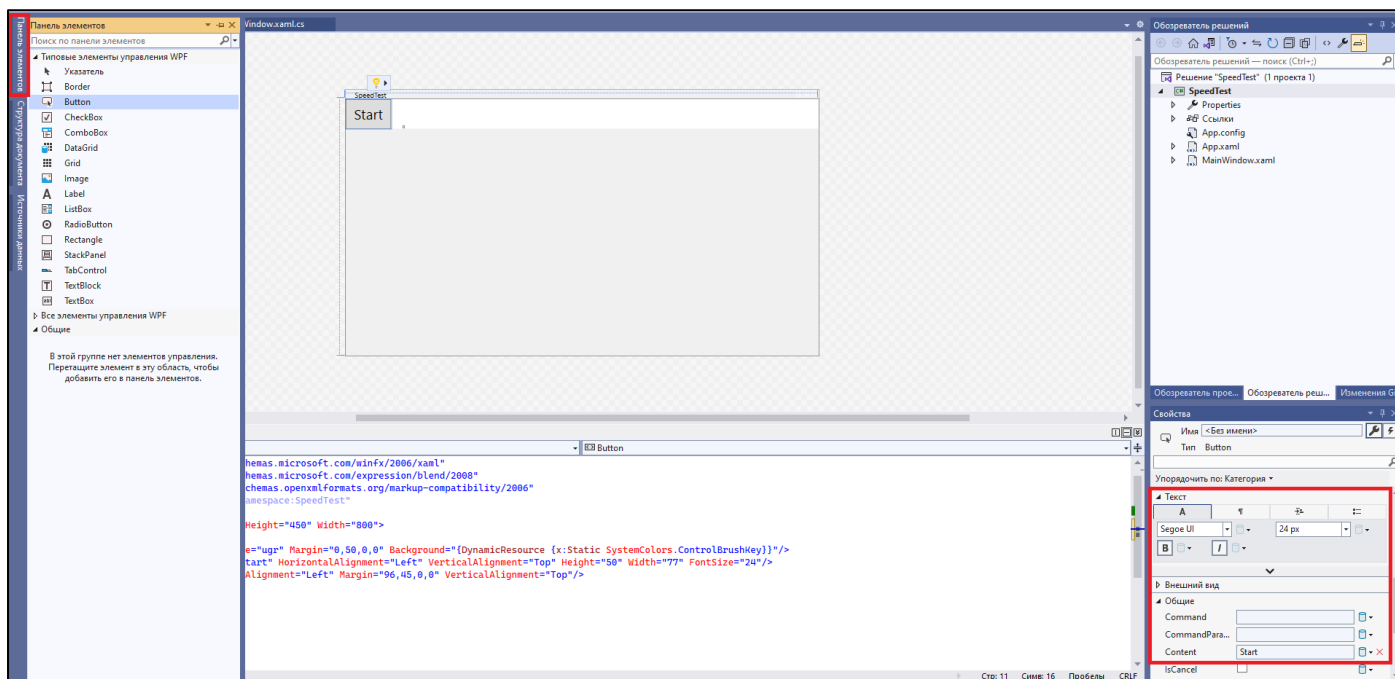
В рамках разрабатываемого проекта, вам понадобится создать игровое поле, кнопку запуска, поле для отображения прошедшего времени и поле для отображения числа открытых элементов игрового поля.

Игровое поле можно создать, используя компонент UniformGrid. Для этого впишите `<UniformGrid>` в XAML редакторе, а затем, задайте имя и фон этого компонента в окне свойств:



Примечание: фон можно выбрать в разделе “Кисть”.

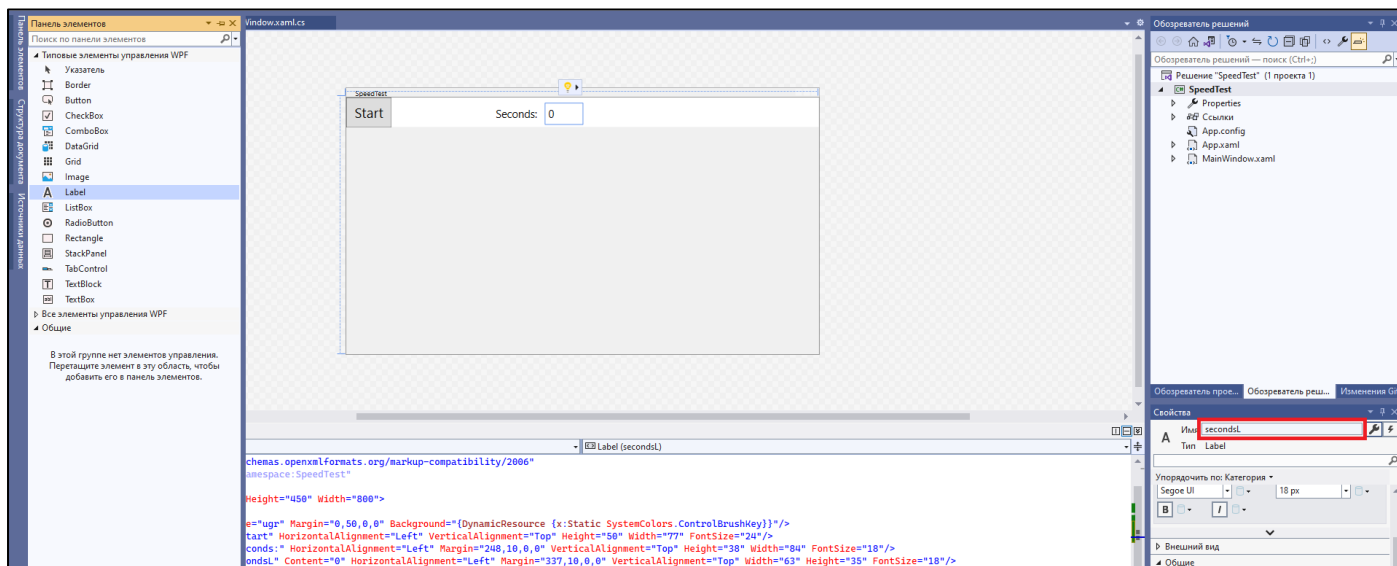
Добавить кнопку можно кликнув на список “Панель элементов” и перетащив компонент Button на форму. Выбрав кнопку на форме, в окне свойств можно задать её текст и настроить шрифт:



В дальнейшем, компонент “Button”, при нажатии, будет вызывать программный код, выполняющий определённые действия, описанные в файле “MainWindow.xaml.cs”.

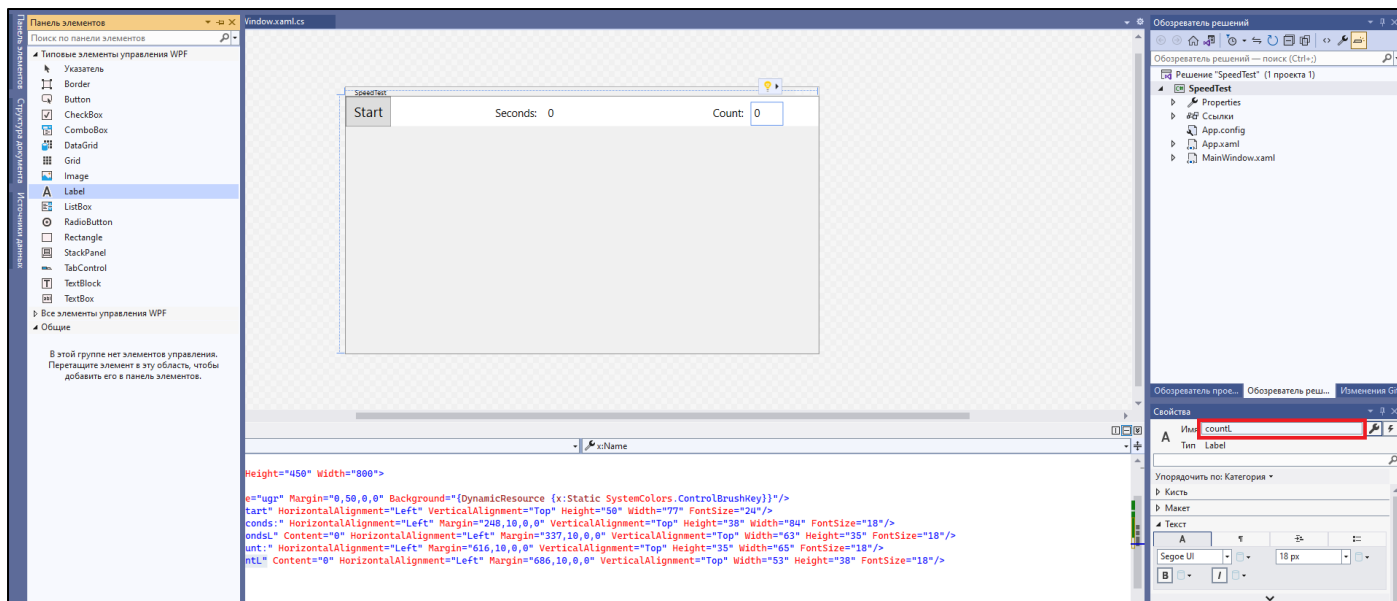
Обратите внимание, что каждый раз, когда вы меняете параметры компонента в окне свойств, соответствующие изменения происходят в XAML редакторе. Это же верно и в обратную сторону, вы можете прописать изменения в XAML редакторе, и они будут автоматически внесены в окне свойств. Таким образом, вы можете вносить изменения любым удобным для вас способом.

Используя список “Панель элементов”, добавьте два компонента типа Label как показано на изображении ниже:



Компоненты типа “Label” используются для отображения информации. В данном случае, первый компонент будет являться подписью, а второй будет служить для отображения информации, полученной в ходе работы программы. Для того, чтобы иметь возможность обратиться ко второму компоненту, ему необходимо задать имя через окно свойств или XAML редактор.

По аналогии с предыдущей парой компонентов типа Label, создайте подпись и поле для отображения числа открытых элементов:



Часть 2: программный код

Программный код приложения находится в файле “MainWindow.xaml.cs”.

Размер поля можно задать, используя переменную целочисленного типа (int). Переменная, по сути, является именем некой области памяти, в которой будут храниться данные определённого типа. Поскольку размер поля не может быть дробным числом, имеет смысл хранить его в переменной, предназначенной для хранения целых чисел.

В языке C#, для создания переменной, необходимо указать её тип и имя в формате: *тип имя*;

Оператор “;” означает завершение операции. В случае необходимости, созданной переменной можно сразу же присвоить значение при помощи оператора “=”. Пример: `int number = 5;` Где `int` – тип, `number` – имя и `5` – значение.

Используемые в предыдущем разделе компоненты, помещённые на форму, содержат множество различных переменных. Обратиться к ним можно по имени компонента, как показано на изображении ниже:

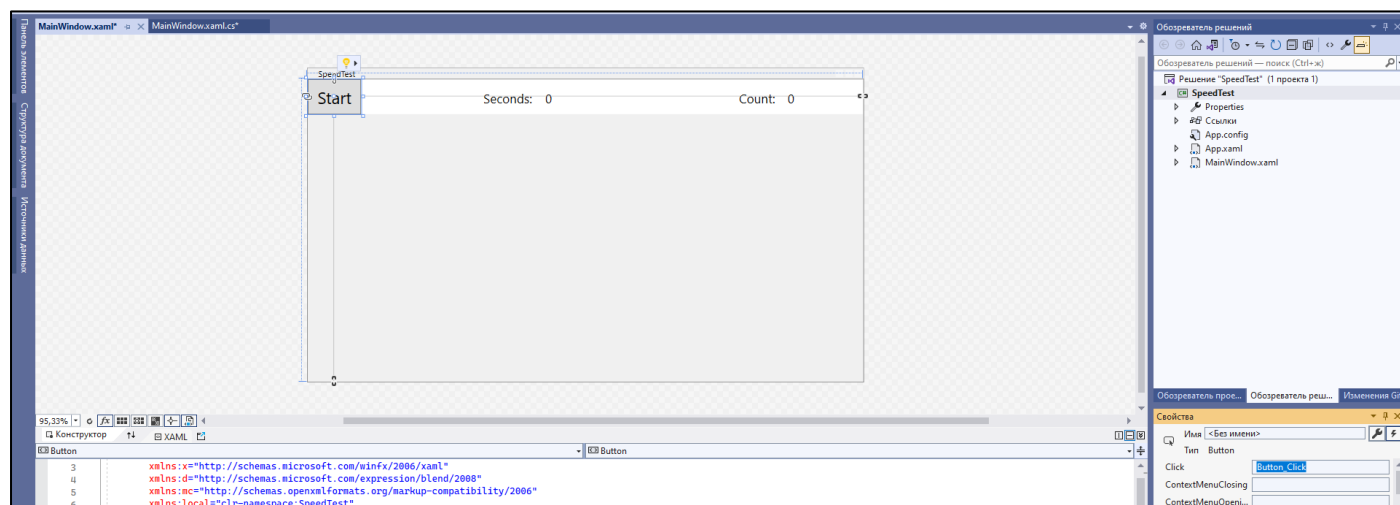
```
namespace SpeedTest
{
    /// <summary>
    /// Логика взаимодействия для MainWindow.xaml
    /// </summary>
    Ссылка: 2
    public partial class MainWindow : Window
    {
        int N = 5; //размерность поля

        Ссылка: 0
        public MainWindow()
        {
            InitializeComponent();

            //указывается количество строк и столбцов в сетке
            ugr.Rows = N;
            ugr.Columns = N;
            //указываются размеры сетки (число ячеек * (размер кнопки в ячейки + толщина её границ))
            ugr.Width = N * (50 + 4);
            ugr.Height = N * (50 + 4);
            //толщина границ сетки
            ugr.Margin = new Thickness(5, 5, 5, 5);
        }
    }
}
```

После того, как вы задали размерность поля и настроили параметры отображения компонента `UniformGrid`, можно переходить к созданию обработчика события нажатия на кнопку “Start”.

Создать обработчик события можно либо дважды кликнув по кнопке, либо выбрав кнопку, нажав значок молнии в окне свойств и дважды кликнув напротив поля “Click”:



Если всё было сделано правильно, в файле, содержащем исходный код, должен появиться подобный метод:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    // ...
}
```

Программный код, описанный в этом методе будет выполняться при нажатии кнопки.

Для того, чтобы осуществить генерацию игрового поля, необходимо создать $N \times N$ кнопок и добавить их в компонент `UniformGrid`. Сделать это можно при помощи циклического оператора (цикла) `for`.

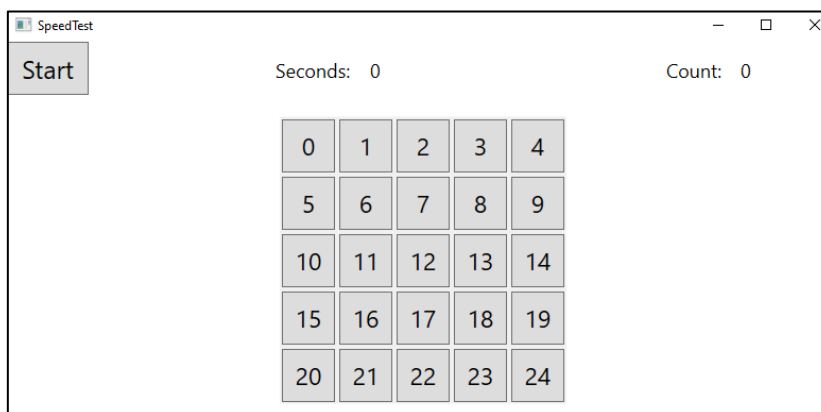
Циклический оператор `for` имеет следующую структуру: *for(блок объявления переменных; условие окончания цикла; блок изменения переменных)*. Операция, или блок операций (исходный код между

символами “{” и “}”), указанные после циклического оператора будут повторяться до тех пор, пока не выполнится условие его остановки.

Компонент Button является переменной и может быть создан, изменён и добавлен в компонент UniformGrid как показано на изображении ниже:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    for (int i = 0; i < N * N; i++)
    {
        //создание кнопки
        Button btn = new Button();
        //запись номера кнопки
        btn.Tag = i;
        //установка размеров кнопки
        btn.Width = 50;
        btn.Height = 50;
        btn.FontSize = 23;
        //текст на кнопке
        btn.Content = i;
        //толщина границ кнопки
        btn.Margin = new Thickness(2);
        //добавление кнопки в сетку
        ugr.Children.Add(btn);
    }
}
```

Если всё было сделано правильно, то при запуске приложения и нажатии кнопки “Start”, на экране должно появиться игровое поле, заполненное числами от 0 до 24:



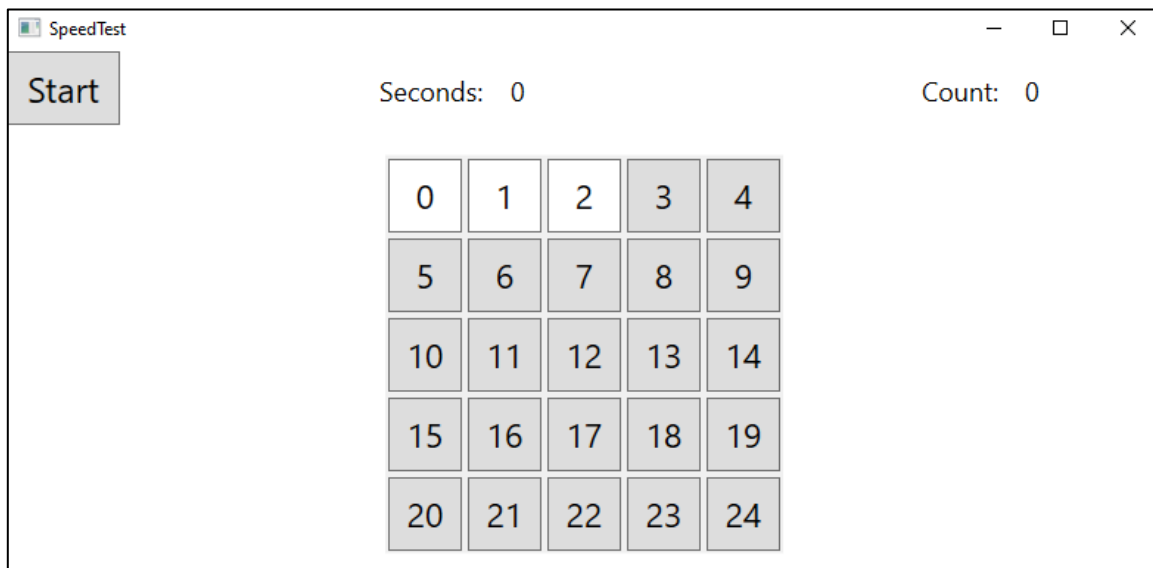
Следующим этапом разработки приложения будет добавление обработчика события нажатия кнопок игрового поля. Метод, срабатывающий при нажатии на кнопку может быть добавлен в одно из полей кнопки как показано на изображении ниже:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    for (int i = 0; i < N * N; i++)
    {
        //создание кнопки
        Button btn = new Button();
        //запись номера кнопки
        btn.Tag = i;
        //установка размеров кнопки
        btn.Width = 50;
        btn.Height = 50;
        btn.FontSize = 23;
        //текст на кнопке
        btn.Content = i;
        //толщина границ кнопки
        btn.Margin = new Thickness(2);
        //при нажатии кнопки, будет вызываться метод Btn_Click
        btn.Click += Btn_Click;
        //добавление кнопки в сетку
        ugr.Children.Add(btn);
    }
}

private void Btn_Click(object sender, RoutedEventArgs e)
{
    //установка цвета фона нажатой кнопки
    ((Button)sender).Background = Brushes.White;
}
```

Примечание: sender – является ссылкой на нажатую кнопку. То есть метод будет изменять цвет фона нажатой кнопки.

Если всё было сделано правильно, то после запуска приложения, нажатия кнопки старт и кликов по элементам поля, их цвет должен изменяться:



После создания игрового поля и реализации взаимодействия с его элементами, можно приступить к реализации игровой механики. Для этого понадобится создать пару массивов, содержащих в себе значения элементов игрового поля, переменную, содержащую количество открытых элементов поля и генератор псевдослучайных чисел.

Массив, это последовательность переменных, имеющих одно и то же имя. В случае, если требуется хранить несколько логически связанных значений, вместо создания множества отдельных переменных, удобнее создать массив. Обращаться к элементам массива можно по их номерам (индексам). Пример:

Создание массива из десяти элементов: `int[] m = new int[10];`

Запись в пятый элемент массива: `m[5] = 3;`

Получение значения из третьего элемента массива: `int n = m[3];`

Поиск номеров, идущих по порядку был бы слишком простой задачей, поэтому предлагается использовать генератор псевдослучайных чисел, для создания восходящей последовательности чисел, идущих не по порядку.

В языке C#, за псевдослучайные последовательности отвечает класс `Random`. Использовать его можно создав переменную соответствующего типа:

```
public partial class MainWindow : Window
{
    int N = 5; //размерность поля
    int[] original; //игровое поле до перемешивания
    int[] field; //игровое поле после перемешивания
    int count = 0; //счётчик открытых полей
    Random rng = new Random(); //генератор псевдослучайных чисел
}
```

Получить случайное значение в диапазоне можно вызвав метод `Next`: `int n = rng.Next(1,5);` В данном случае, в целочисленную переменную с именем “n”, будет записано случайное значение в диапазоне от 1 до 5 (1, 2, 3 или 4).

Так как в дальнейшем, элементы игрового поля будут перемешаны, понадобится два массива. Один, содержащий оригинальную последовательность чисел, для проверки корректности последовательности нажатия пользователем кнопок игрового поля. Второй, для отображения перемешанных значений.

В целом, метод обработки нажатия кнопки “Start” будет выглядеть следующим образом:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    count = 0; //обнуление счётчика открытых полей

    //выделение памяти под поля
    field = new int[N * N];
    original = new int[N * N];

    int n = 1; //стартовое значение
    for (int i = 0; i < N * N; i++)
    {
        original[i] = field[i] = n; //запись значения в игровые поля
        n += rng.Next(1, 5); //смещение значения на случайную величину
    }

    for (int i = 0; i < N * N; i++)
    {
        //создание кнопки
        Button btn = new Button();
        //запись номера кнопки
        btn.Tag = field[i];
        //установка размеров кнопки
        btn.Width = 50;
        btn.Height = 50;
        btn.FontSize = 23;
        //текст на кнопке
        btn.Content = field[i];
        //толщина границ кнопки
        btn.Margin = new Thickness(2);
        //при нажатии кнопки, будет вызываться метод Btn_Click
        btn.Click += Btn_Click;
        //добавление кнопки в сетку
        ugr.Children.Add(btn);
    }
}
```

Поскольку теперь в программе есть массив, содержащий корректную последовательность нажатия кнопок игрового поля, в метод обработки события нажатия можно добавить условия проверки корректности нажатия и победы.

В языке C#, условный оператор имеет следующий if (если) формат: if (условие) {если истина} else {если ложь}. В данном случае, номер нажатой кнопки совпадает с номером, который нужно открыть в текущий момент, то будет увеличиваться число открытых элементов игрового поля, обновляться информация о количестве открытых элементов, изменяться фон нажатой кнопки и выполняться проверка на победу:

```
private void Btn_Click(object sender, RoutedEventArgs e)
{
    //получение значения лежащего в Tag
    int n = (int)((Button)sender).Tag;

    if (original[count] == n) //если номер нажатой кнопки совпадает с следующим по порядку
    {
        count++; //увеличить число открытых полей
        countL.Content = count; //обновить число открытых полей на форме

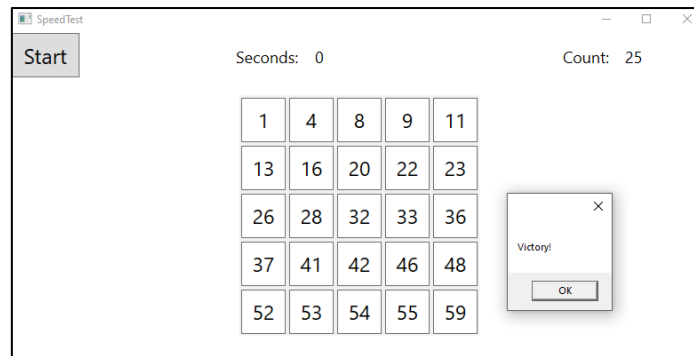
        //установка фона нажатой кнопки, цвета и размера шрифта
        ((Button)sender).Background = Brushes.White;

        if (count == N * N) //если число открытых полей равно общему числу полей
        {
            MessageBox.Show("Victory!"); //вывести сообщение о победе
            ugr.Children.Clear(); //очистить игровое поле
        }
    }
}
```

В случае, если число открытых элементов совпадёт с общим числом элементов, будет выведено сообщение о победе и произведена очистка игрового поля.

Для вывода сообщения используется стандартный класс MessageBox, который создаёт и показывает окно, содержащее сообщение, указанное в скобках метода Show.

Если всё было сделано корректно, то при последовательном нажатии всех кнопок на игровом поле должно появиться сообщение о победе:



Перемешивание элементов в массиве будет состоять из трёх этапов:

1. выбрать два случайных элемента в массиве
2. поменять местами выбранные элементы
3. проверить, сколько элементов осталось на своих местах

Реализовать эти этапы удобнее всего при помощи функций. Функция, по сути, является небольшой частью программы, которую можно выполнять множество раз, обратившись к ней по её имени.

В С# функции имеют следующий формат: *возвращаемое_значение имя_функции(список_параметров)*

Например, функция возвращающая результат сложения двух переменных может быть описана как: `int add(int a, int b);` При этом вызываться она будет как: `int c = add(1, 2);` //c – будет содержать 3

Описать функцию обмена значениями двух переменных можно как:

```
void swap(ref int a, ref int b) //обмен значениями двух переменных через третью
{
    int t = a;
    a = b;
    b = t;
}
```

Примечание: параметры функциями являются копиями передаваемых значений, чтобы изменить сами передаваемые значения, нужно использовать ключевое слово `ref`

Функция, определяющая сколько элементов поля, осталось на своих местах:

```
bool check() //проверка, сколько процентов элементов игрового поля осталось на своих местах
{
    double k = 0; //число элементов поля оставшихся на своих местах

    for (int i = 0; i < N * N; i++)
    {
        if (original[i] == field[i]) k++; //если элемент игрового поля до и после перемешивания находится на том же месте
    }

    k /= N * N; //вычисление процента элементов, оставшихся на своих местах

    if (k < 0.2) //если на своих местах осталось менее 20% элементов – возвращается Истина
        return true;
    else return false;
}
```

Функция смешивания будет обменивать местами случайно выбранные элементы массива, до тех пор, пока не будет пройдена проверка:

```
void mixer() //перемешивание элементов игрового поля
{
    while (check() != true)
    {
        int a = rng.Next(N * N);
        int b = rng.Next(N * N);
        swap(ref field[a], ref field[b]);
    }
}
```

Обратите внимание, что вместо циклического оператора `for` использованного ранее, функция `mixer` использует циклический оператор `while`. Оператор `for` удобен в использовании, когда заранее известно

количество повторений операций, находящихся внутри цикла, в данном же случае, известно только условие окончания, сколько повторений случится до выполнения этого условия – не известно. Формат циклического оператора `while` выглядит как: `while(условие_остановки) {тело_цикла}` Существует вариация `while`, в которой условие проверяется после выполнения цикла, однако в этой работе она не используется: `do {тело_цикла} while(условие_остановки);`

Последним этапом будет модификация обработчика нажатия кнопки “Start”:

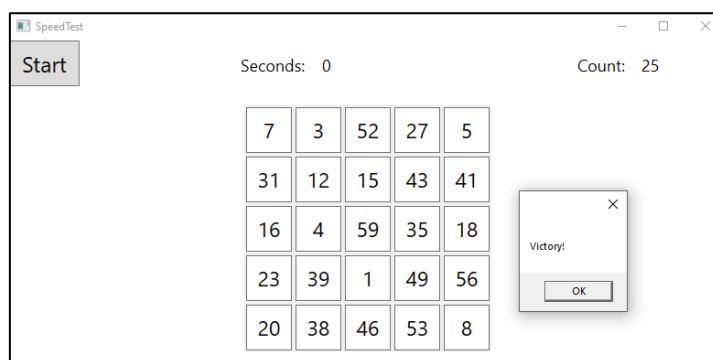
```
private void Button_Click(object sender, RoutedEventArgs e)
{
    count = 0; //обнуление счётчика открытых полей
    ugr.Children.Clear(); //очистка игрового поля
    countL.Content = count; //обнуление счётчика открытых полей

    //выделение памяти под поля
    field = new int[N * N];
    original = new int[N * N];

    int n = 1; //стартовое значение

    for (int i = 0; i < N * N; i++)
    {
        mixer(); //перемешивание игрового поля
    }
}
```

Результат внесённых изменений должен быть похож на изображение ниже:



Завершающим этапом разработки будет добавление и настройка объекта “Таймер”:

```
Random rng = new Random(); //генератор псевдослучайных чисел

int seconds = 0; //счётчик секунд
System.Windows.Threading.DispatcherTimer dt; //таймер

Ссылка: 0
public MainWindow()
{
    InitializeComponent();

    //указывается количество строк и столбцов в сетке
    ugr.Rows = N;
    ugr.Columns = N;
    //указываются размеры сетки (число ячеек * (размер кнопки в ячейки + толщина её границ))
    ugr.Width = N * (50 + 4);
    ugr.Height = N * (50 + 4);
    //толщина границ сетки
    ugr.Margin = new Thickness(5, 5, 5, 5);

    dt = new System.Windows.Threading.DispatcherTimer(); //создание таймера
    //назначение обработчика события Тик
    dt.Tick += Dt_Tick;
    //установка интервала между тиками
    //TimeSpan – переменная для хранения времени в формате часы/минуты/секунды
    dt.Interval = new TimeSpan(0, 0, 1);
}

Ссылка: 1
private void Dt_Tick(object sender, EventArgs e)
{
    seconds++;
    secondsL.Content = seconds.ToString() + " sec";
}
```

Таймер, это стандартный компонент, не имеющий визуального отображения. Для работы с ним, необходимо задать интервал времени, через который будет вызываться некая функция, а также вызываемую функцию. Поскольку в данном приложении таймер будет просто отсчитывать время, интервал имеет смысл выставить в одну секунду, а в функции увеличивать значение переменной, содержащее количество прошедших секунд и обновлять информацию о прошедшем времени на форме.

В обработчик нажатия кнопки “Start” следует добавить обнуление счётчика секунд и запуск таймера:

```
private void Button_Click(object sender, RoutedEventArgs e)
{
    count = 0; //обнуление счётчика открытых полей
    ugr.Children.Clear(); //очистка игрового поля
    countL.Content = count; //обнуление счётчика открытых полей
    seconds = 0; //обнуление счётчика секунд
    dt.Start(); //запуск таймера
}
```

Завершающим этапом разработки будет остановка таймера в случае победы и увеличение счётчика секунд в случае выбора не корректного элемента игрового поля:

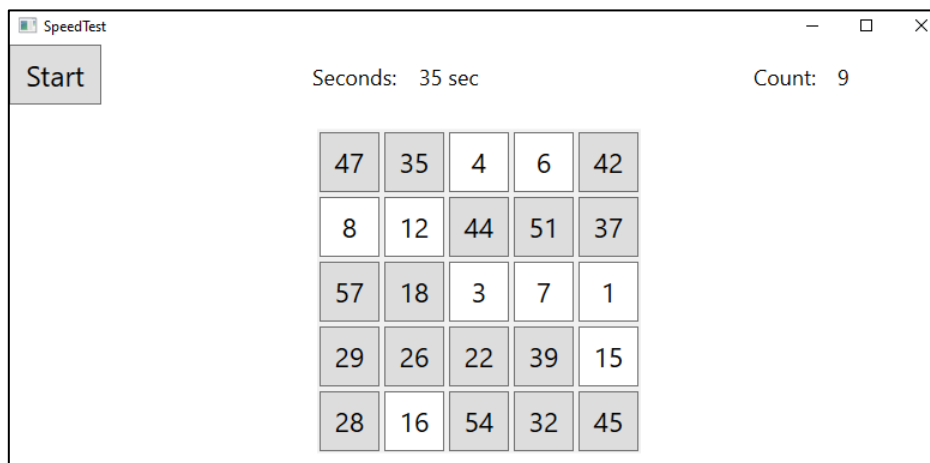
```
private void Btn_Click(object sender, RoutedEventArgs e)
{
    //получение значения лежащего в Tag
    int n = (int)((Button)sender).Tag;

    if (original[count] == n) //если номер нажатой кнопки совпадает с следующим по порядку
    {
        count++; //увеличить число открытых полей
        countL.Content = count; //обновить число открытых полей на форме

        //установка фона нажатой кнопки, цвета и размера шрифта
        ((Button)sender).Background = Brushes.White;

        if (count == N * N) //если число открытых полей равно общему числу полей
        {
            MessageBox.Show("Victory!"); //вывести сообщение о победе
            ugr.Children.Clear(); //очистить игровое поле
            dt.Stop();
        }
        else seconds++;
    }
}
```

Итоговый результат:



Самостоятельные модификации:

При желании, программу можно доработать и/или усложнить одним из следующих способов:

1. изменить условие победы таким образом, чтобы элементы игрового поля нужно было нажимать в нисходящей последовательности (от максимального к минимальному)
2. чтобы при нажатии неверного элемента приходилось начинать сначала
3. что бы после нажатия на каждый элемент поле перемешивалось заново