Beispielprojekt: Mastermind

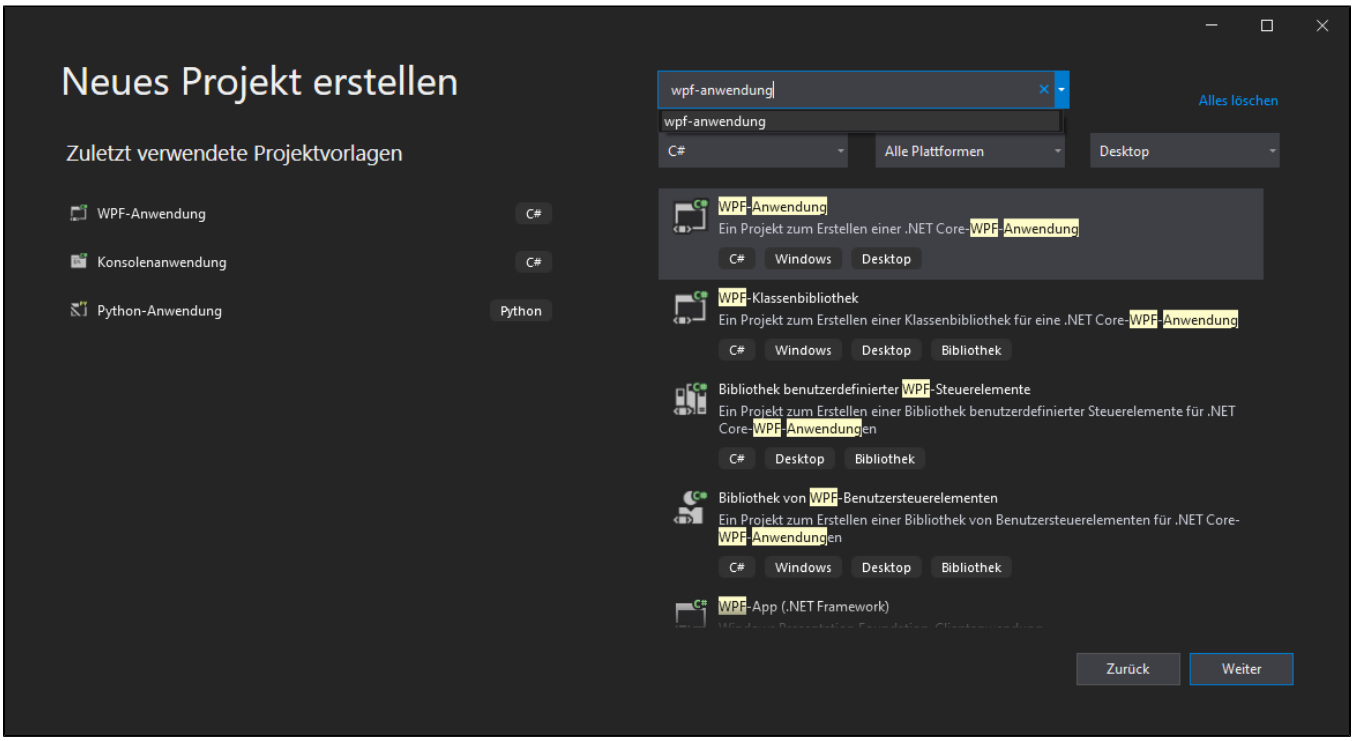
Erstellen einer Neuen WPF-Anwendung

Um eine neue WPF-Anwendung zu erstellen, öffnen Sie zunächst "Visual Studio". Im Anschluss daran navigieren Sie zum Menüpunkt "Neues Projekt erstellen".

In der oberen Suchleiste geben Sie "WPF-Anwendung" ein und wählen die entsprechende Option für C# aus.

Nach diesem Schritt verleihen Sie dem Projekt einen aussagekräftigen Namen (z.B. MasterMind) und klicken auf Weiter.

Wählen Sie im nächsten Dialogfenster das Framework ".NET 7.0 (Standard-Laufzeitunterstützung)" aus und setzen Sie den Erstellungsprozess fort.



Erstellen des XAML-Code

Das erstellen der Benutzeroberfläche:

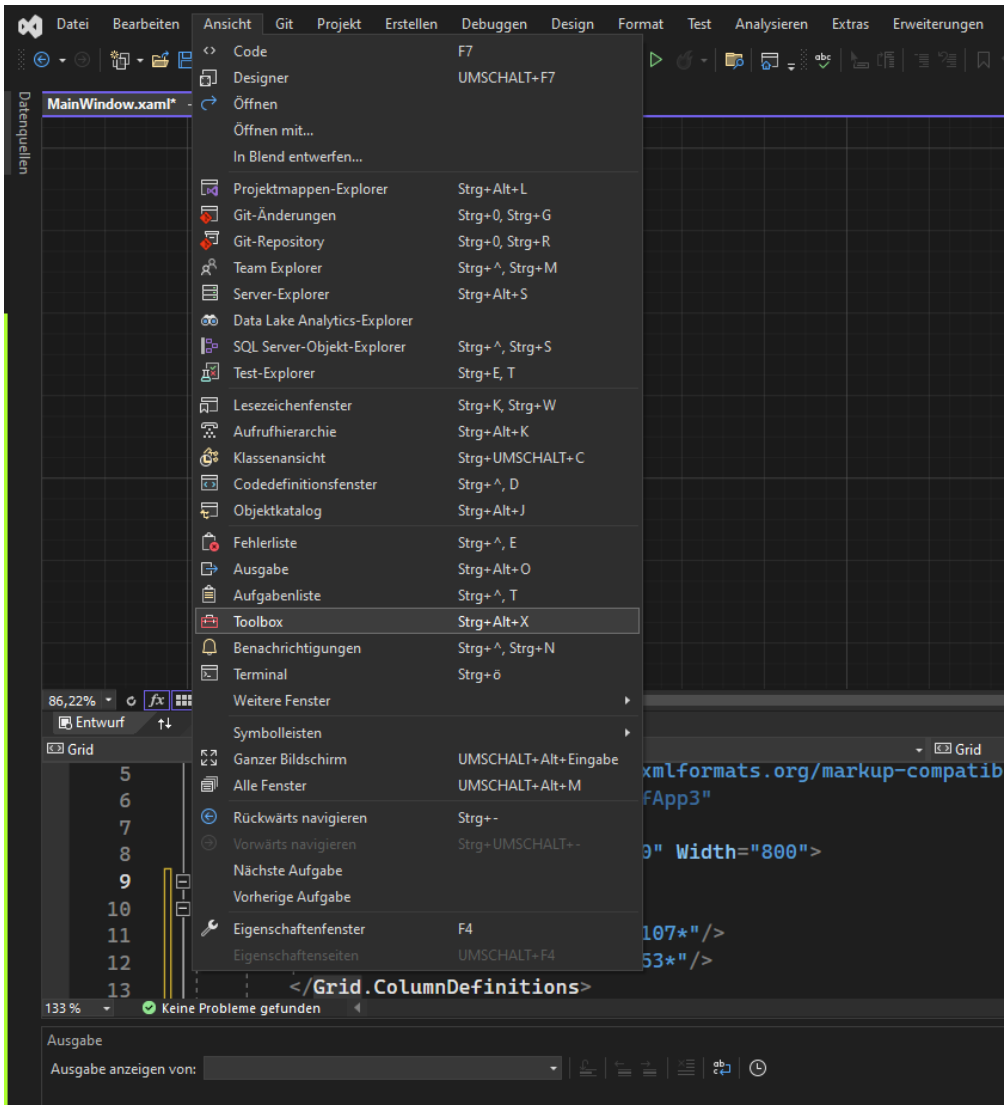
Für die Benutzeroberfläche Brauchen wir einen Button mit dem man die Eingabe Bestätigen kann (Bestätigen), einen mit dem

man neu starten kann (Neues Spiel), eine TextBox zur Eingabe, ListBox für die Ausgabe der Eingabe d(en Tipps) und eine ListBox für die

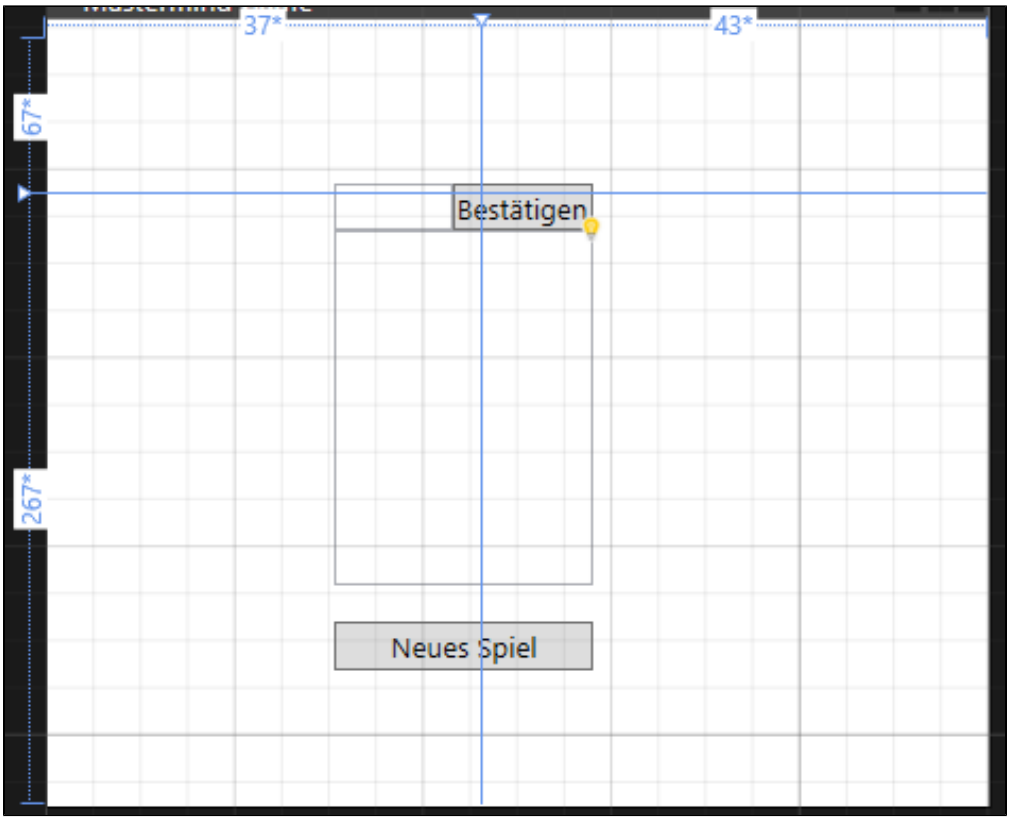
Ausgabe der Übrigen Versuche, wenn man verliert wird dort auch der richtige Code angezeigt .

Um Buttons, TextBox und ListBox zu finden, gehen sie auf die ToolBox, entweder auf der Linken Seite oder unter Ansicht, darunter können sie

die Steuerelemente in der Suchleiste eingeben und mit Doppelklick einfügen:



Als nächstes müssen sie die Steuerelemente in einer Sinnvollen Anordnung individuell hinrichten, eine Logische Anordnung wäre das:


37*
43*
67*
267*
Bestätigen
Neues Spiel

XAML-Code

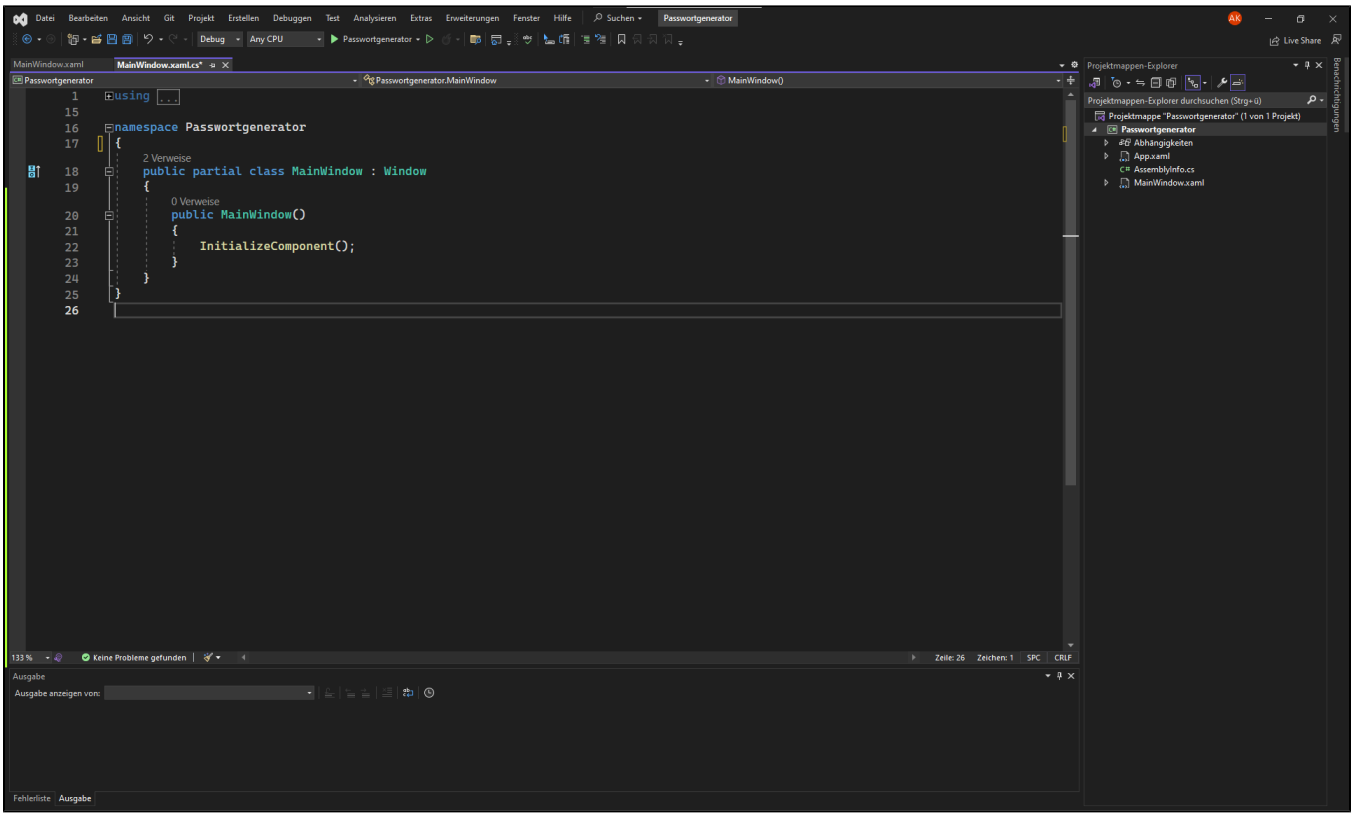
```
<Window x:Class="MastermindGame.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Mastermind Game" Height="350" Width="400">
    <Grid Margin="0,10,0,0" HorizontalAlignment="Center" Width="400" Height="324" VerticalAlignment="Top">
        <Grid.RowDefinitions>
            <RowDefinition Height="67*"/>
            <RowDefinition Height="267*"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="37*"/>
            <ColumnDefinition Width="43*"/>
        </Grid.ColumnDefinitions>
        <StackPanel Margin="122,60,168,58" Grid.ColumnSpan="2" Grid.RowSpan="2">

            <WrapPanel>
                <TextBox x:Name="GuessInput" Width="50" />
                <Button Content="Bestätigen" Click="GuessButton_Click" />
            </WrapPanel>


            <ListBox x:Name="GuessHistory" Height="150">

                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <TextBlock Text="{Binding}" />
                    </DataTemplate>
                </ListBox.ItemTemplate>
            </ListBox>


            <TextBlock x:Name="ResultText" FontWeight="Bold" Foreground="Green" />


            <Button Content="Neues Spiel" Click="NewGameButton_Click" />
        </StackPanel>
    </Grid>
</Window>
```

Code Behind

Unser Programm ist derzeit in einem rudimentären Zustand. Um es zu verbessern, fügen wir nun unseren Code ein.

Navigiert dazu entweder über den Tab "MainWindow.xaml.cs" oder benutzt die Tastenkombination Strg + F6, um zum "Code-Behind" zu gelangen.

Zuerst deklarieren wir die benötigten Variablen.



Anschließend verleihen wir unseren Knöpfen eine Funktion, um eine sinnvolle Interaktion zu ermöglichen.

Durch diese Schritte verbessern wir die Funktionalität und Bedienbarkeit unseres Programms erheblich.

Klasse "MainWindow":

- Die Hauptklasse des Programms, die von der "Window"-Klasse erbt und somit ein WPF-Fenster darstellt.

Felddeklarationen:

- Deklariert private Felder, um den geheimen Code und die Anzahl der verbleibenden Versuche zu speichern.
- Mit kann man "string" speichert man Zeichenkettene
- Mit "int" kann man Zahlen speichern im Bereich von 2.147.483.648 bis 2.147.483.647

Deklarieren

```
        private string secretCode;
        private int VersucheÜbrig = 12;
```

Methoden

Konstruktor `MainWindow()und Neustart()-Methode:`

- Der Konstruktor initialisiert das Hauptfenster und ruft die "Neustart()"-Methode auf, um ein neues Spiel zu starten.
- "Neustart()" setzt verschiedene Spielvariablen zurück und generiert einen neuen geheimen Code.

Konstruktor

```
        public MainWindow()
        {
            InitializeComponent();
            Neustart();
        }
        private void Neustart()
        {
            secretCode = GeheimCode();
            GuessHistory.Items.Clear();
            ResultText.Text = string.Empty;
            VersucheÜbrig = 12;
       }
```

"GuessButton_Click"Methode:

- Wird aufgerufen, wenn der Benutzer auf den "Rate"-Button klickt.
- Überprüft die Eingabe des Benutzers, aktualisiert die Spielhistorie und gibt Feedback zum eingegebenen Code.

GuessButton

```
private void GuessButton_Click(object sender, RoutedEventArgs e)
{
        if (VersucheÜbrig > 0)
    {
            string Bestätigen = GuessInput.Text.Trim();

        if (Bestätigen.Length == 4 && Bestätigen.All(char.IsDigit))
        {
                string result = CheckGuess(Bestätigen);
            GuessHistory.Items.Insert(0, $"Bestätigen: {Bestätigen} Info: {result}");
            GuessInput.Clear();
            VersucheÜbrig--;
        }
        if (result == "1111")
        {
                ResultText.Text = "Das war der Richtige code!";
            GuessInput.IsEnabled = false;
        }
        else if (VersucheÜbrig == 0)
        {
                        ResultText.Text = ($"Du hast verloren der code wäre {secretCode} gewessen");
        }
        else
        {
                ResultText.Text = $"Versuche Übrig: {VersucheÜbrig}";
        }
        else
        {
                        ResultText.Text = "Eingabe nicht möglich. Es musss eine Nummer mit 4 Stellen sein!";
        }
     }
}
```

CheckGuess()-Methode:

- Überprüft den vom Benutzer eingegebenen Code und gibt ein Feedback zurück, das angibt, wie viele Ziffern an der richtigen Position und wie viele an der falschen Position sind.

CheckGuss

```
private string CheckGuess(string Bestätigen)
{
        int correctPosition = secretCode.Where((c, i) => c == Bestätigen[i]).Count();
        int inWrongPosition = secretCode.Count(c => Bestätigen.Contains(c)) - correctPosition;
        return new string('1', correctPosition) + new string('2', inWrongPosition);
}
```

"NewGameButton_Click"Methode:

- Wird aufgerufen, wenn der Benutzer auf den "Neues Spiel"-Button klickt.
- Startet ein neues Spiel, setzt die Variablen zurück und ermöglicht dem Benutzer, erneut zu raten.

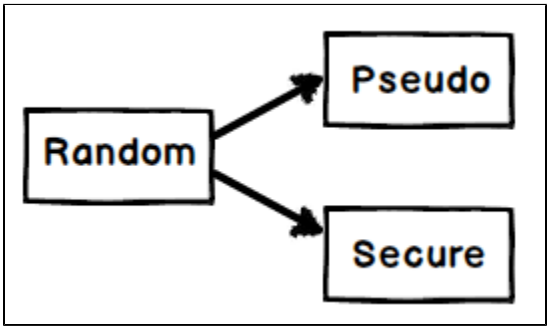
NewGameButton

```
private void NewGameButton_Click(object sender, RoutedEventArgs e)
{
        Neustart();
        GuessInput.IsEnabled = true;
}
```

Erstellen der Funktion für die Zufallsgeneration:

Was ist eigentlich Zufall in der Programmierung?

Es gibt kein richtiges Random in der Programmierung sondern es gibt zwei arten von Random es gibt "Pseudo-" und es gibt "Secure-" random.

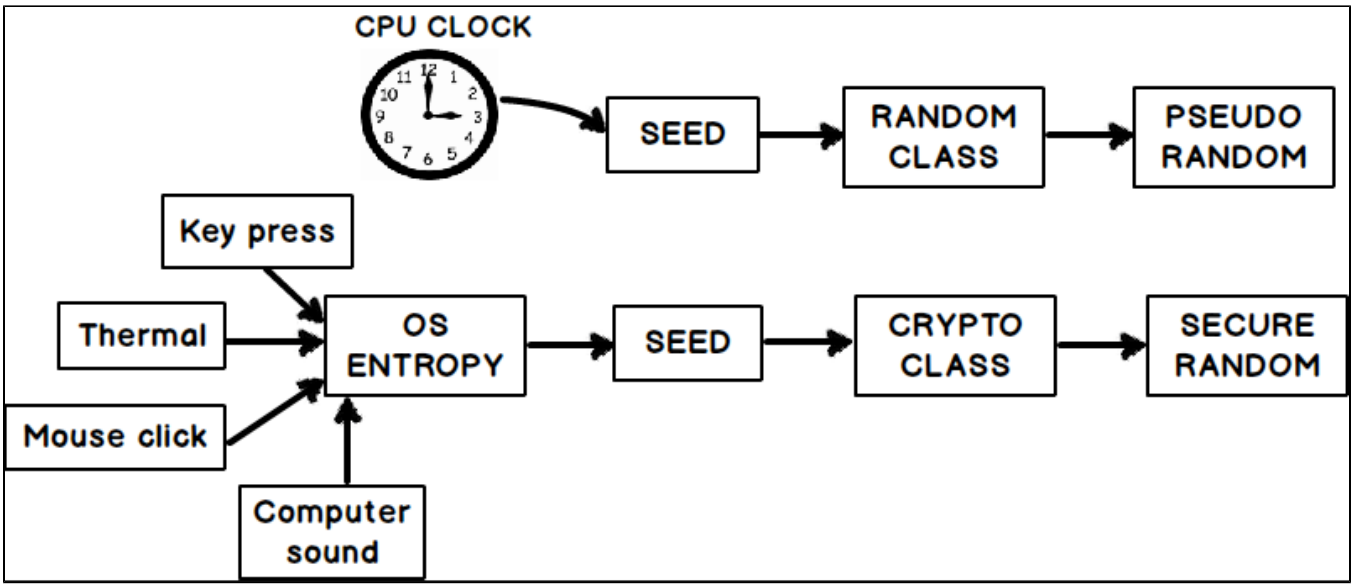


Um etwas zufällig zu gestalten, bedarf es eines Trigger (Seed). Bei Menschen entsteht dieser Seed in Bruchteilen einer Sekunde.

Bei der Frage nach Grün, Rot oder Gelb spielen zahlreiche Faktoren eine Rolle, wie zum Beispiel die Lieblingsfarbe oder die Glücksfarbe.

Im Fall eines Computers wird der Seed, wie in unserem Beispiel, mithilfe der Random-Klasse über die CPU-Uhr generiert.

Für unser Beispiel ist dies vollkommen ausreichend, jedoch ist in bestimmten Spielen oder anderen Anwendungen eine sichere Zufallszahlengenerierung erforderlich. Dies könnte durch die Verwendung der "RNGCryptoServiceProvider"-Klasse sichergestellt werden, die beispielsweise für Verschlüsselungszwecke verwendet wird.



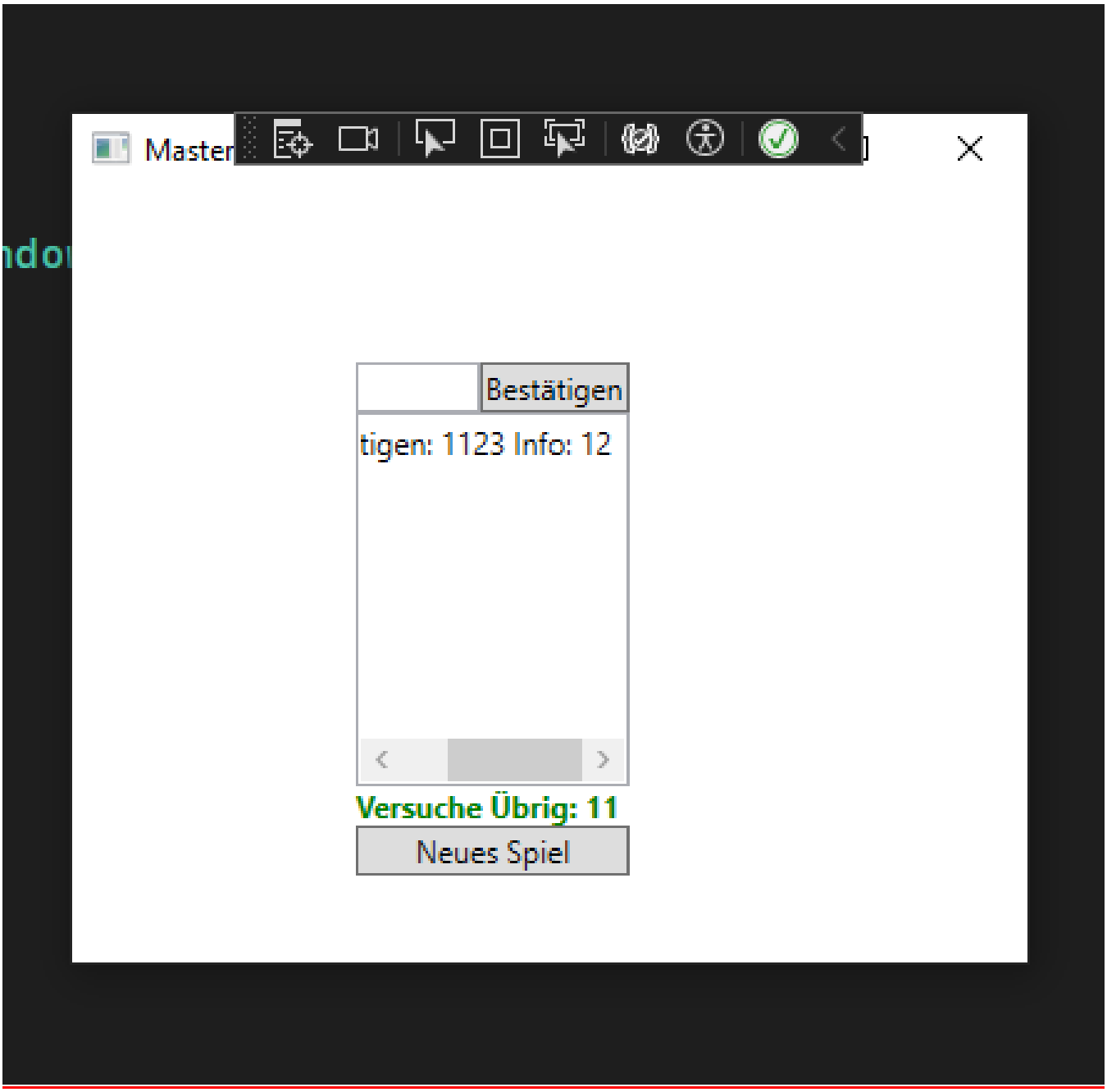
Programmierung der Zufalls Methode:

- Generiert einen zufälligen geheimen Code, der aus vier Ziffern besteht.

GeheimCode

```
private string GeheimCode()
{
        Random random = new Random();
        return string.Join("", Enumerable.Range(0, 4).Select(_ => random.Next(1, 7)));
}
```

Wenn du mit allem fertig bist sollte dein Programm so aussehen:



Fertig

```
using System;
using System.Linq;
using System.Windows;

namespace MastermindGame
{
    public partial class MainWindow : Window
    {
        private string secretCode;
        private int VersucheÜbrig = 12;
```

```
        public MainWindow()
        {
            InitializeComponent();
            Neustart();
        }
        private void Neustart()
        {
            secretCode = GeheimCode();
            GuessHistory.Items.Clear();
            ResultText.Text = string.Empty;
            VersucheÜbrig = 12;
        }

        private string GeheimCode()
        {

            Random random = new Random();
            return string.Join("", Enumerable.Range(0, 4).Select(_ => random.Next(1, 7)));
        }
        private void GuessButton_Click(object sender, RoutedEventArgs e)
        {
            if (VersucheÜbrig > 0)
            {
                string Bestätigen = GuessInput.Text.Trim();
                if (Bestätigen.Length == 4 && Bestätigen.All(char.IsDigit))
                {
                    string result = CheckGuess(Bestätigen);
                    GuessHistory.Items.Insert(0, $"Bestätigen: {Bestätigen} Info: {result}");
                    GuessInput.Clear();
                    VersucheÜbrig--;

                    if (result == "1111")
                    {
                        ResultText.Text = "Das war der Richtige code!";
                        GuessInput.IsEnabled = false;
                    }
                    else if (VersucheÜbrig == 0)
                    {
                        ResultText.Text = ($"Du hast verloren der code wäre {secretCode} gewessen");
                    }
                    else
                    {
                        ResultText.Text = $"Versuche Übrig: {VersucheÜbrig}";
                    }
                }
                else
                {
                    ResultText.Text = "Eingabe nicht möglich. Es musss eine Nummer mit 4 Stellen sein!";
                }
            }
        }
        private string CheckGuess(string Bestätigen)
        {
            int correctPosition = secretCode.Where((c, i) => c == Bestätigen[i]).Count();

            int inWrongPosition = secretCode.Count(c => Bestätigen.Contains(c)) - correctPosition;

            return new string('1', correctPosition) + new string('2', inWrongPosition);
        }

        private void NewGameButton_Click(object sender, RoutedEventArgs e)
        {
            Neustart();
            GuessInput.IsEnabled = true;
        }
    }
}
```