Beispielprojekt: Checkboxgame

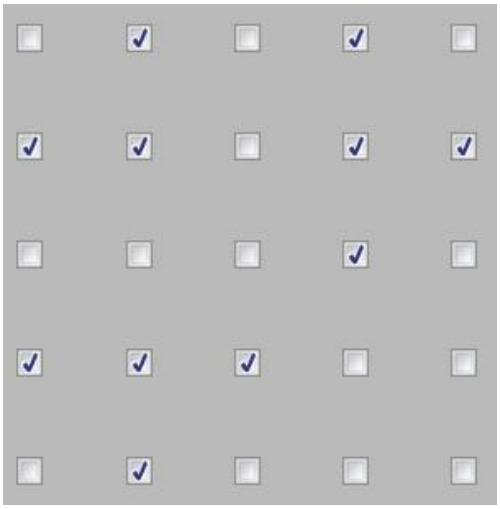
Wer Professor Layton mag, ist mit diesem Projekt gut beraten. Checkboxgame ist ein kleines Rätselspiel, dass, selbst wenn man den Algorithmus dahinter kennt, versteht und das Spiel programmiert hat, dennoch einige Köpfe zum Rauchen bringen wird. Ein sehr simples und einfaches Spielkonzept, sowie ein sehr einfacher Code machen dieses Spiel besonders Anfängerfreundlich.

Inhaltsverzeichnis / list of contents

Ein erstes Spiel

Das Konzept

Ziel des Spiels ist es, in einem 5x5 Feld aus Checkboxen, alle Haken zu setzen. Was die Schwierigkeit dabei ist? Natürlich hat die Sache einen Haken 🙂. Wenn man in einen Haken setzt oder entfernt, soll sich der Haken in den umliegenden Checkboxen (oben, unten, links und rechts) ebenfalls in das jeweils entgegengesetzte ändern. D.h., dass wenn man bei folgendem Feld:



in die Checkbox in der linken oberen Ecke klicken würde, würde die Ecke den Status "checked" annehmen und die unmittelbar rechte und untere Checkbox den Status "unchecked".

Die graphische Oberfläche

Nach dieser kleinen Vorüberlegung kann man eigentlich schon loslegen:

Man öffnet Visual Studio und erstellt ein neues Projekt unter das links oben in der Ecke befindliche: **Datei Neu Projekt WPF-Anwendung**

Bitte wählt auch einen geeigneten Projektnamen für eure Anwendung!

Da die Arbeit mit der graphischen Oberfläche ziemlich unspannend ist, werde ich den xaml-code zur Verfügung stellen. Bitte beachtet dabei, dass ihr den Projektnamen an den geeigneten Stellen im xaml-code einfügt. Ich werde diese Stellen mit "PROJEKTNAME_HIER_EINFUEGEN" markieren.

(natürlich könnt ihr auch selbst herumprobieren, wenn ihr möchtet 🙂, allerdings solltet ihr dabei einige Dinge beachten. Dazu später mehr.)

XAML-Code

```
<Window x:Name="Fenster" x:Class="PROJEKTNAME_HIER_EINFUEGEN.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:PROJEKTNAME_HIER_EINFUEGEN"
        mc:Ignorable="d"
        Title="Check all Checkboxes" Height="350" Width="300" Background="#FFBABABA" Loaded="Fenster_Loaded">
    <Grid x:Name="Grid">
        <CheckBox x:Name="c0" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
35,26,0,0" Width="16" Height="16" Click="c0_Click"/>
        <CheckBox x:Name="c1" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
35,78,0,0" Width="16" Height="16" Click="c1_Click" />
        <CheckBox x:Name="c2" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
35,130,0,0" Width="16" Height="16" Click="c2_Click" />
        <CheckBox x:Name="c3" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
35,182,0,0" Width="16" Height="16" Click="c3_Click" />
        <CheckBox x:Name="c4" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
35,234,0,0" Width="16" Height="16" Click="c4_Click" />
        <CheckBox x:Name="c5" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
88,26,0,0" Width="16" Height="16" Click="c5_Click"/>
        <CheckBox x:Name="c6" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
88,78,0,0" Width="16" Height="16" Click="c6_Click" />
        <CheckBox x:Name="c7" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
88,130,0,0" Width="16" Height="16" Click="c7_Click" />
        <CheckBox x:Name="c8" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
88,182,0,0" Width="16" Height="16" Click="c8_Click" />
        <CheckBox x:Name="c9" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
88,234,0,0" Width="16" Height="16" Click="c9_Click" />
        <CheckBox x:Name="c10" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
140,26,0,0" Width="16" Height="16" Click="c10_Click"/>
        <CheckBox x:Name="c11" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
140,78,0,0" Width="16" Height="16" Click="c11_Click" />
        <CheckBox x:Name="c12" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
140,130,0,0" Width="16" Height="16" Click="c12_Click" />
        <CheckBox x:Name="c13" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
140,182,0,0" Width="16" Height="16" Click="c13_Click" />
        <CheckBox x:Name="c14" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
140,234,0,0" Width="16" Height="16" Click="c14_Click" />
        <CheckBox x:Name="c15" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
192,26,0,0" Width="16" Height="16" Click="c15_Click" />
        <CheckBox x:Name="c16" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
192,78,0,0" Width="16" Height="16" Click="c16_Click" />
        <CheckBox x:Name="c17" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
192,130,0,0" Width="16" Height="16" Click="c17_Click" />
        <CheckBox x:Name="c18" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
192,182,0,0" Width="16" Height="16" Click="c18_Click" />
        <CheckBox x:Name="c19" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
192,234,0,0" Width="16" Height="16" Click="c19_Click" />
        <CheckBox x:Name="c20" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
244,26,0,0" Width="16" Height="16" Click="c20_Click" />
        <CheckBox x:Name="c21" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
244,78,0,0" Width="16" Height="16" Click="c21_Click" />
        <CheckBox x:Name="c22" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
244,130,0,0" Width="16" Height="16" Click="c22_Click" />
        <CheckBox x:Name="c23" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
244,182,0,0" Width="16" Height="16" Click="c23_Click" />
        <CheckBox x:Name="c24" Content="CheckBox" HorizontalAlignment="Left" VerticalAlignment="Top" Margin="
244,234,0,0" Width="16" Height="16" Click="c24_Click" />
        <Button x:Name="button" Content="Reset" HorizontalAlignment="Left" Height="23" Margin="88,278,0,0"
VerticalAlignment="Top" Width="120" Click="button_Click"/>
    </Grid>
</Window>
```

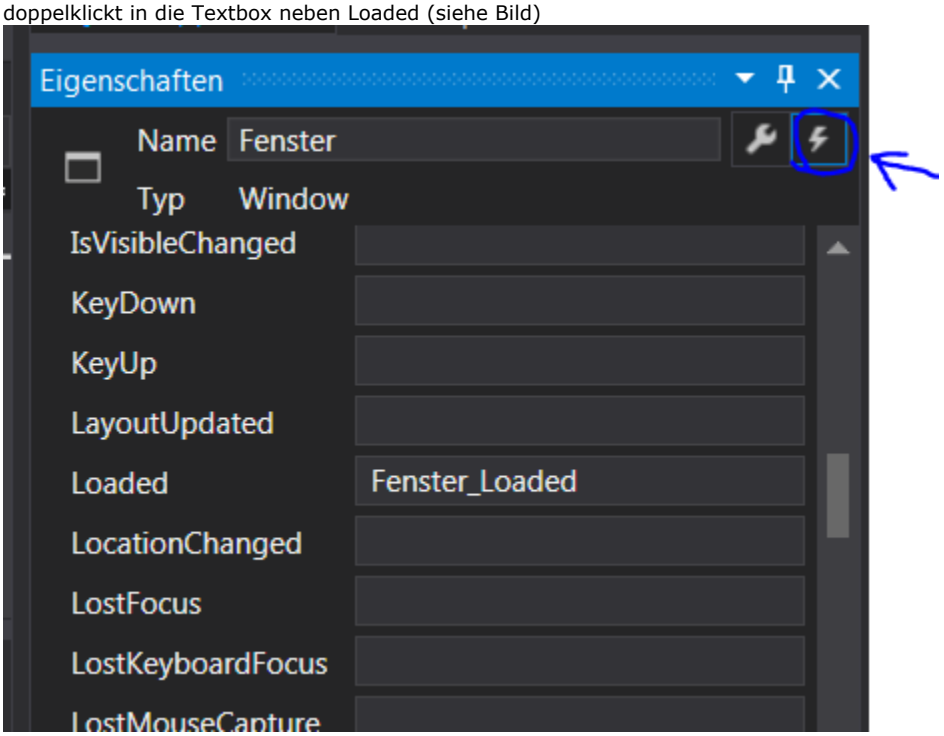
Die Oberfläche sollte nun 25 Checkboxen in einem 5x5 Raster, sowie einen Button "Reset" enthalten.

Nun ist Folgendes zu beachten:

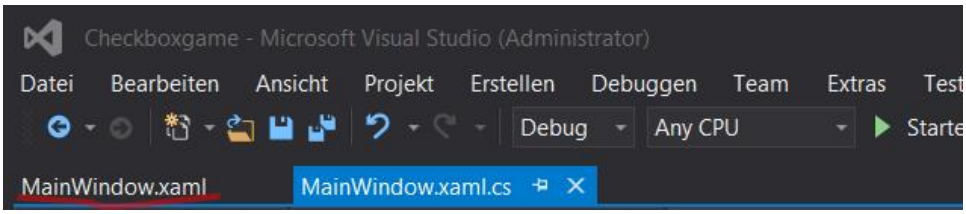
- Der Button, das Fenster und jede einzelne Checkbox besitzen Namen. Ich werde im Folgenden Code **MEINE EIGENEN** Name n verwenden.**Wenn ihr selbst herumprobiert: Jedes Objekt sollte einen Namen besitzen!**. Diese Namen werden folgendermaßen gesetzt:

```
    x:Name="OBJEKTNAME" z.B. x:Name="Fenster" bei der Deklaration des Fensters oder x:Name="c20" bei der
Deklaration der Checkbox 20.
```

- Das Fenster besitzt eine Methode: "Loaded="Fenster_Loaded"" Diese sollte hinzugefügt werden, wenn ihr selbst herumprobiert. Entweder direkt im xaml-Code oder ihr wählt das Fenster an und geht rechts auf das Blitzsymbol und doppelklickt in die Textbox neben Loaded (siehe Bild)



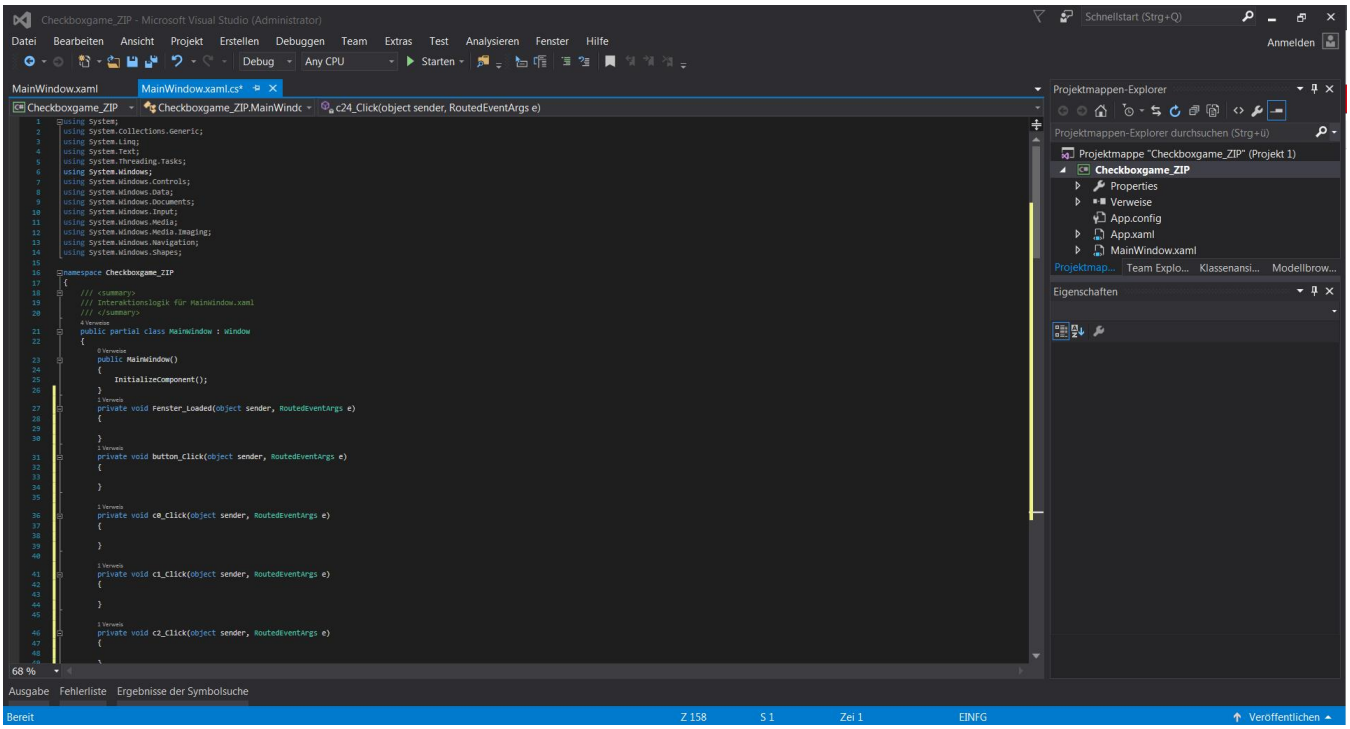
- Die Checkboxen, sowie der Button besitzen eine Methode: "Click="OBJEKTNAME_Click"" diese sollte bei diesen hinzugefügt werden. Dies funktioniert ähnlich, wie bei der "Loaded" Methode, nur dass ihr neben "Click" anstatt "Loaded" doppelklickt. Dies muss leider für jedes Objekt einzeln getan werden...
- Nebenbei, **wenn ihr die Methode mit Doppelklick wählt**: Visual Studio springt anschließend gleich zur entsprechenden Stelle im Code. Um zurück zur graphischen Ansicht zu kommen, könnt ihr ganz einfach oben den Tab zu "MainWindow.xaml" wechseln.



Ordnung des Codes und Überlegungen

Methodendefinition der Oberfläche

Nun wäre die graphische Oberfläche abgeschlossen und ihr solltet, wenn ihr nun zum Code geht (durch einen Tastendruck auf F7) solltet ihr in etwa folgenden Anblick erhalten:



Wenn die Methoden **Fenster_Loaded**, **button_Click** und **cXX_Click** nicht vorhanden sind, sind diese zu ergänzen.

Zu beachten ist hierbei, dass die Methodennamen mit den in der xaml festgelegten übereinstimmen. (Bei meinem Beispiel, steht in der xaml an entsprechender Stelle beispielsweise: **Click="c0_Click"**. Üblicherweise trägt diese Methode den Namen OBJEKTNAME_ACTION (wobei ACTION in diesem Fall "Click" oder "Loaded" bedeutet. Die Parameter dieser Funktionen müssen mit die des Bildes identisch sein.)

Zu beachten ist Außerdem, dass ihr die Click Methoden ALLER Checkboxen einfügt. (Auf dem Bild ist dafür zu wenig Platz)

Der Konstruktor der Klasse "MainWindow" ist zwar meist nützlich, aber in diesem Programm unnötig und kann problemlos entfernt werden.

Die Klasse ECheck

Nun zu den Überlegungen:

Um das gewünschte Ergebnis zu bekommen, sollten wir eine weitere Klasse erstellen, welche jede der 25 Checkboxen "erweitert", d.h. jeder Checkbox die 4 umliegenden zuordnet. Ich nenne diese Klasse geeigneterweise "ECheck". Einem Objekt der Klasse ECheck wird bei der Erstellung im Konstruktor eine Checkbox übergeben, welche erweitert werden soll. Desweiteren soll das Objekt die vier umliegenden Checkboxen als Attribute bokommen, die vorerst nur deklariert und nicht initialisiert werden. Außerdem wird noch eine Checkbox "th" deklariert, die mit der im Konstruktor übergebenen Checkbox initialisiert wird.

Klasse "ECheck" mit Konstruktor

```
        public class ECheck
    {
        public CheckBox re, li, ob, un, th; //Die 4 umliegenden Checkboxen (re,li,ob,un), sowie die zu
erweiternde Checkbox (th) werden deklariert

        public ECheck(CheckBox t) // Konstruktor der Klasse ECheck, dem die zu erweiternde Checkbox
übergeben wird
        {
            th = t; //th wird mit t initialisiert
        }
    }
```

Um die umliegenden Checkboxen zu initialisieren brauchen wir eine Methode "Integrate", der die 4 Checkboxen übergeben werden. Anschließend wird noch eine Methode "Change" benötigt, die den Status "Checked" der umliegenden Checkboxen umkehrt, falls diese existieren.

"Integrate" und "Change"

```
                public void Integrate(CheckBox l, CheckBox r, CheckBox o, CheckBox u) //Methode zur
Initialisierung der 4 umliegenden Checkboxen
        {
            li = l; //linke Checkbox wird initialisiert
            re = r; //rechte Checkbox wird initialisiert
            ob = o; //obere Checkbox wird initialisiert
            un = u; //untere Checkbox wird initialisiert
        }
        public void Change() //Methode zum umkehren der Stati der umliegenden Checkboxen
        {
            if(li!=null) //Wenn linke Checkbox existiert
            {
                li.IsChecked = !li.IsChecked;             //kehre den Status dieser Checkbox um
            }
            if (re != null) //Wenn rechte Checkbox existiert
            {
                re.IsChecked = !re.IsChecked;                 //...
            }
            if (un != null) //...
            {
                un.IsChecked = !un.IsChecked; //...
            }
            if (ob != null) //...
            {
                ob.IsChecked = !ob.IsChecked; //...
            }
        }
```

Damit wären wir mit der Klasse ECheck fertig und können nun zur Klasse MainWindow übergehen:

Die Klasse MainWindow

Zunächst braucht die Klasse MainWindow als Attribut ein Feld der Klasse ECheck, welches in der Methode Fenster_Loaded initialisiert wird. Anschließend wird in der gleichen Methode jeder erweiterten Checkbox, die umliegenden Checkboxen mit der Methode "Integrate" übergeben, sodass diese initialisiert werden können. Dies ist eine sehr hakelige Aufgabe, bei der schnell Fehler passieren können. Also achtet darauf.

Methode Fenster_Loaded

```
        public partial class MainWindow : Window
    {
        public ECheck[] ec = new ECheck[25]; //Feld von Erweiterten Checkboxen als Attribut der Klasse
MainWindow

        private void Fenster_Loaded(object sender, RoutedEventArgs e)
        {
            ec[0] = new ECheck(c0); //Die zu erweiternden Checkboxen werden der Klasse ECheck bei der
Erstellung des Objekts übergeben.
            ec[1] = new ECheck(c1);
            ec[2] = new ECheck(c2);
            ec[3] = new ECheck(c3);
            ec[4] = new ECheck(c4);
            ec[5] = new ECheck(c5);
            ec[6] = new ECheck(c6);
            ec[7] = new ECheck(c7);
            ec[8] = new ECheck(c8);
            ec[9] = new ECheck(c9);

            ec[10] = new ECheck(c10);
            ec[11] = new ECheck(c11);
            ec[12] = new ECheck(c12);
            ec[13] = new ECheck(c13);
            ec[14] = new ECheck(c14);
            ec[15] = new ECheck(c15);
            ec[16] = new ECheck(c16);
            ec[17] = new ECheck(c17);
            ec[18] = new ECheck(c18);
            ec[19] = new ECheck(c19);

            ec[20] = new ECheck(c20);
            ec[21] = new ECheck(c21);
            ec[22] = new ECheck(c22);
            ec[23] = new ECheck(c23);
            ec[24] = new ECheck(c24);


            ec[0].Integrate(null, c1, null, c5); //Die 4 umliegenden Checkboxen werden den Objekten der
Klasse ECheck durch die Methode Integrate übergeben.
            ec[1].Integrate(c0, c2, null, c6);
            ec[2].Integrate(c1, c3, null, c7);
            ec[3].Integrate(c2, c4, null, c8);
            ec[4].Integrate(c3, null, null, c9);
            ec[5].Integrate(null, c6, c0, c10);
            ec[6].Integrate(c5, c7, c1, c11);
            ec[7].Integrate(c6, c8, c2, c12);
            ec[8].Integrate(c7, c9, c3, c13);
            ec[9].Integrate(c8, null, c4, c14);

            ec[10].Integrate(null, c11, c5, c15);
            ec[11].Integrate(c10, c12, c6, c16);
            ec[12].Integrate(c11, c13, c7, c17);
            ec[13].Integrate(c12, c14, c8, c18);
            ec[14].Integrate(c13, null, c9, c19);
            ec[15].Integrate(null, c16, c10, c20);
            ec[16].Integrate(c15, c17, c11, c21);
            ec[17].Integrate(c16, c18, c12, c22);
            ec[18].Integrate(c17, c19, c13, c23);
            ec[19].Integrate(c18, null, c14, c24);

            ec[20].Integrate(null, c21, c15, null);
            ec[21].Integrate(c20, c22, c16, null);
            ec[22].Integrate(c21, c23, c17, null);
            ec[23].Integrate(c22, c24, c18, null);
            ec[24].Integrate(c23, null, c19, null);
        }
```

Der Reset-Button der Klasse MainWindow soll alle Checkboxen zurück auf Status "unchecked" setzen. Dafür muss man nur mit einer for-Schleife für jedes "th"-Attribut jedes ECheck-Objekts des Feldes ec auf "unchecked" setzen.

Reset-Button

```
                private void button_Click(object sender, RoutedEventArgs e)
        {
            for (int i = 0; i < ec.Length; ++i) //Lasse i von 0-LängeDesFeldes_ec-1 laufen und erhöhe i nach
jedem Schritt um 1
            {
                ec[i].th.IsChecked = false; //Setze Status, der "th"-Checkbox des i-ten Elements des Feldes
ec auf Status "unchecked"
            }
        }
```

Schön und gut. Aber wann gewinnt man? Wenn jeder Haken gesetzt wurde. Wie frägt man dies am Besten ab? Ich schlage vor, den Status der Checkbox "th" eines jeden Elements in in ec nacheinander bei jedem Klick auf eine Checkbox abzufragen und falls eine Checkbox dabei sein sollte, welche "unchecked" ist, die Methode abzubrechen. Am Ende der Methode (wenn keine Checkbox gefunden wurde, welche "uncheckt" ist) kann anschließend eine Siegesmeldung ausgegeben werden.

Siegesmethode

```
        public void Win()
        {
            for (int i = 0; i < ec.Length; ++i) //Schleife, die jedes Element des Feldes überprüft
            {
                if (!(bool)ec[i].th.IsChecked) //Welchen Status hat die Checkbox? (mit Typecast, da der
zurückgegebene bool nullable ist
                {
                    return; //Methodenabbruch
                }
            }
            MessageBox.Show("Gewonnen!"); //Siegesmeldung
        }
```

Okay, nun haben wir fast alles, was wir brauchen. Jetzt bleibt nur eines: Was soll bei Klick auf die Checkbox passieren? Hier wird es leider etwas eintönig, da wir alle CHECKBOXNAME_Click Methoden einzeln befüllen müssen... Glücklicherweise ist der Code, der in jede Klick-Methode muss, nicht sonderlich lang. 🙂

Checkboxen

```
        private void c0_Click(object sender, RoutedEventArgs e)
        {
            ec[0].Change(); //Der Status wird geändert
            Win();                        //Überprüfung, ob man gesiegt hat
        }

        private void c1_Click(object sender, RoutedEventArgs e)
        {
            ec[1].Change(); //...
            Win();                        //...
        }

        private void c2_Click(object sender, RoutedEventArgs e)
        {
            ec[2].Change();
            Win();
        }

        private void c3_Click(object sender, RoutedEventArgs e)
        {
            ec[3].Change();
            Win();
        }
```

```
private void c4_Click(object sender, RoutedEventArgs e)
{
    ec[4].Change();
    Win();
}

private void c5_Click(object sender, RoutedEventArgs e)
{
    ec[5].Change();
    Win();
}

private void c6_Click(object sender, RoutedEventArgs e)
{
    ec[6].Change();
    Win();
}

private void c7_Click(object sender, RoutedEventArgs e)
{
    ec[7].Change();
    Win();
}

private void c8_Click(object sender, RoutedEventArgs e)
{
    ec[8].Change();
    Win();
}

private void c9_Click(object sender, RoutedEventArgs e)
{
    ec[9].Change();
    Win();
}

private void c10_Click(object sender, RoutedEventArgs e)
{
    ec[10].Change();
    Win();
}

private void c11_Click(object sender, RoutedEventArgs e)
{
    ec[11].Change();
    Win();
}

private void c12_Click(object sender, RoutedEventArgs e)
{
    ec[12].Change();
    Win();
}

private void c13_Click(object sender, RoutedEventArgs e)
{
    ec[13].Change();
    Win();
}

private void c14_Click(object sender, RoutedEventArgs e)
{
    ec[14].Change();
    Win();
}

private void c15_Click(object sender, RoutedEventArgs e)
{
    ec[15].Change();
    Win();
}
```

```
        private void c16_Click(object sender, RoutedEventArgs e)
        {
            ec[16].Change();
            Win();
        }

        private void c17_Click(object sender, RoutedEventArgs e)
        {
            ec[17].Change();
            Win();
        }

        private void c18_Click(object sender, RoutedEventArgs e)
        {
            ec[18].Change();
            Win();
        }

        private void c19_Click(object sender, RoutedEventArgs e)
        {
            ec[19].Change();
            Win();
        }

        private void c20_Click(object sender, RoutedEventArgs e)
        {
            ec[20].Change();
            Win();
        }

        private void c21_Click(object sender, RoutedEventArgs e)
        {
            ec[21].Change();
            Win();
        }

        private void c22_Click(object sender, RoutedEventArgs e)
        {
            ec[22].Change();
            Win();
        }

        private void c23_Click(object sender, RoutedEventArgs e)
        {
            ec[23].Change();
            Win();
        }

        private void c24_Click(object sender, RoutedEventArgs e)
        {
            ec[24].Change();
            Win();
        }
```

Nun, wenn man diese Copy-Paste Arbeit Verrichtet hat, ist das Programm auch schon fertig und ausführbereit und der Rätselspaß kann beginnen 🙂