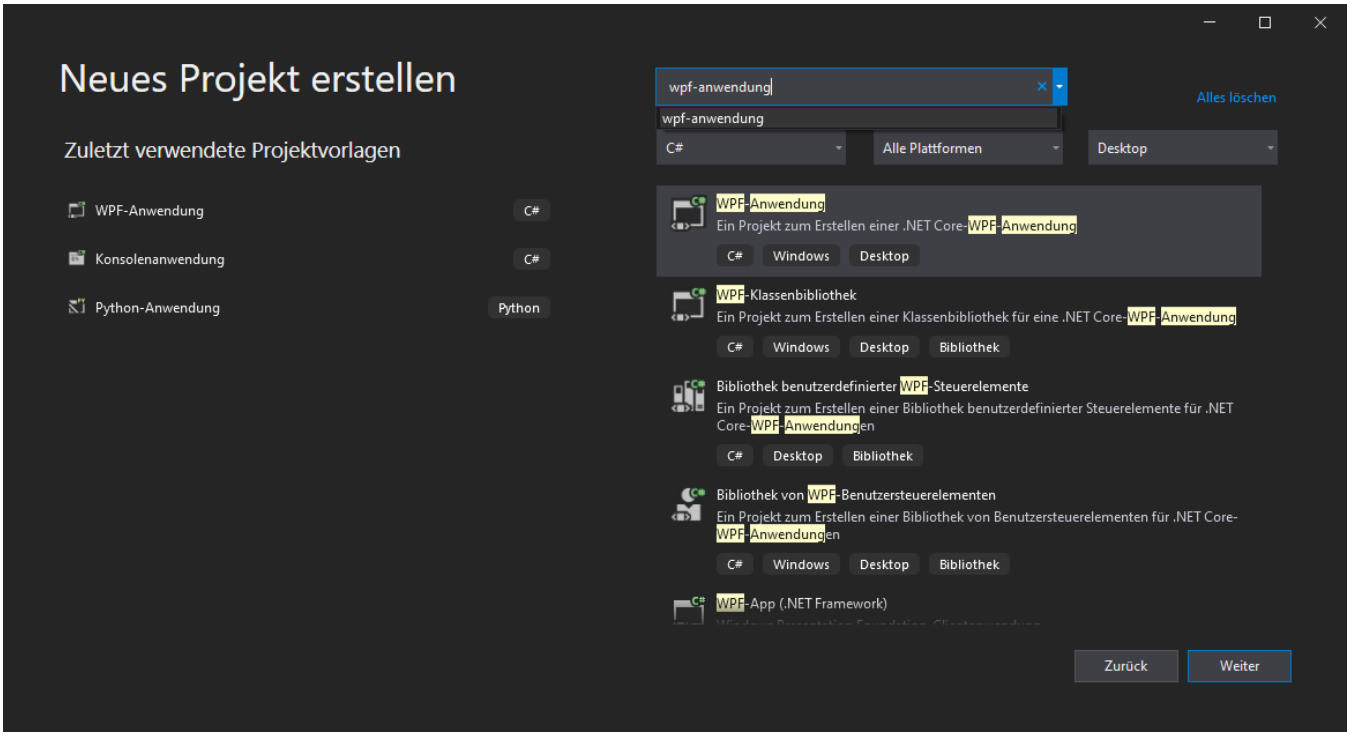
Beispielprojekt: Minesweeper

Visual-Studio-WPF-Projekt für das Game "Minesweeper"

Inhaltsverzeichnis / list of contents

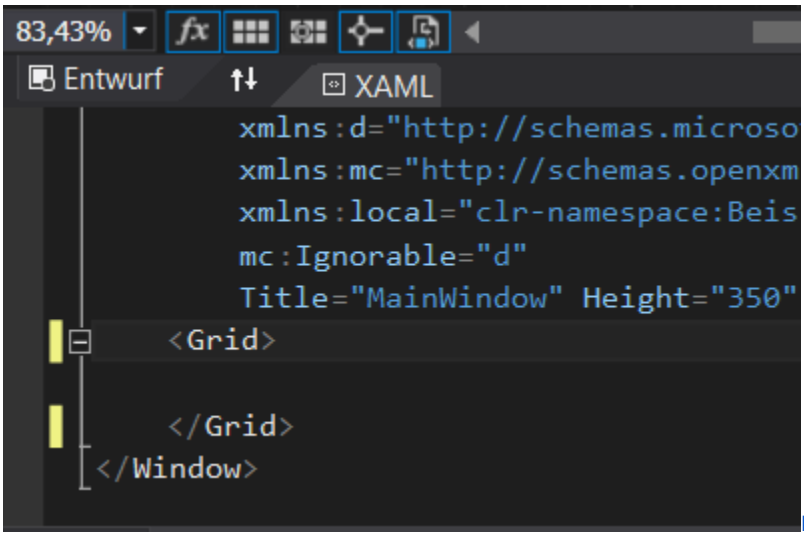
Schritt für Schritt Anleitung

- Neue Wpf-Anwendung in Visual Studio erstellen

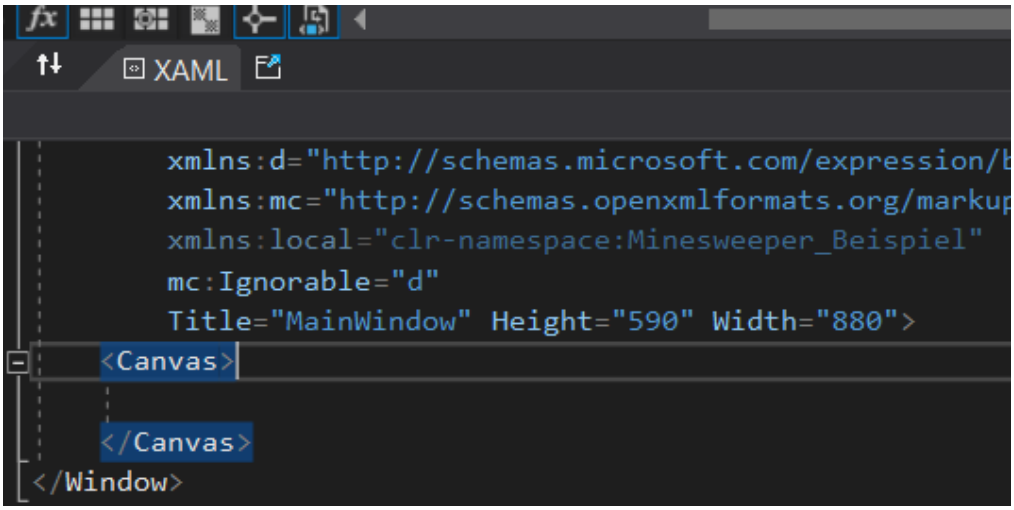


Xaml-Code

- Im Xaml-Editor Grid in Canvas ändern, x(Name) = "canvas" setzen und die Fenster Größe anpassen



blocked URL



MainWindow.xaml

```
<Window x:Class="Minesweeper_Test.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:Minesweeper_Test"
        mc:Ignorable="d"
        Title="MainWindow" Height="590" Width="880">
    <Canvas x:Name="canvas">

    </Canvas>
</Window>
```

Code Behind

Bisher kann unser Programm nicht viel, deshalb fügen wir als nächstes unseren Code in MainWindow.xaml.cs (Shortcut Strg + F6) ein. Dazu müssen wir zunächst:

- Variablen hinzufügen
- eine neue Klasse "GameButtons" und das enum "ButtonState" erstellen
- eine Methode zum Erstellen der Gamebuttons erstellen

- und Methoden mit Spielfunktionen erstellen

Deklarieren der Variablen

Variablen

```
        public int column_count, row_count, seperator, bomb_count, button_dim;
        private int _currentBombCount;
    private readonly Point[] _points = {
        new Point(-1,-1), new Point(0,-1), new Point(1,-1),
        new Point(-1,0),                   new Point(1,0),
        new Point(-1,1),  new Point(0,1),  new Point(1,1)
    };
    private readonly Random _random = new Random();
    private readonly GameButton[,] _fields;
        private TextBlock _bombCountTB;
```

Wir benötigen:

- 6 Variablen des Typs int für Spalten, Reihen, Anzahl der Bomben beim Starten, Bombenzähler im Spiel, den Abstand der Köpfe, und die Größe der Knöpfe
- ein Array der Klasse Point, das mit 8 Punkten gefüllt wird
- eine Zufallszahl
- ein 2 Dimensionales Array der GameButton-Klasse, die im Anschluss implementiert wird
- und ein Objekt der Klasse TextBlock das später die aktuelle Anzahl der Bomben anzeigen soll.

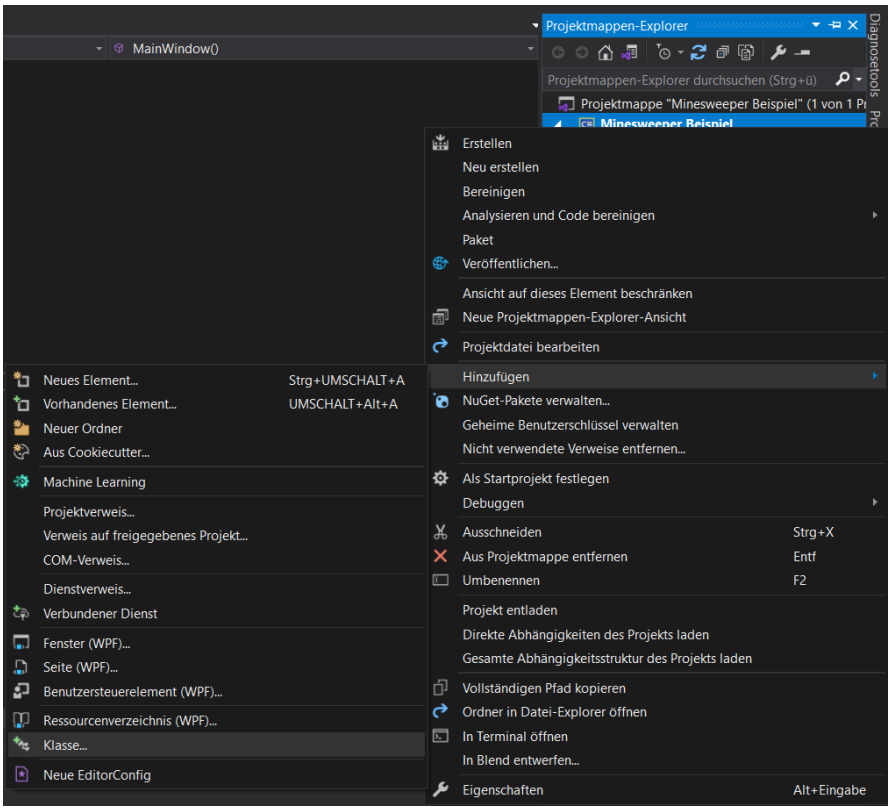
Die Variablen werden im Konstruktor der MainWindow-Klasse mit Werten befüllt

Konstruktor

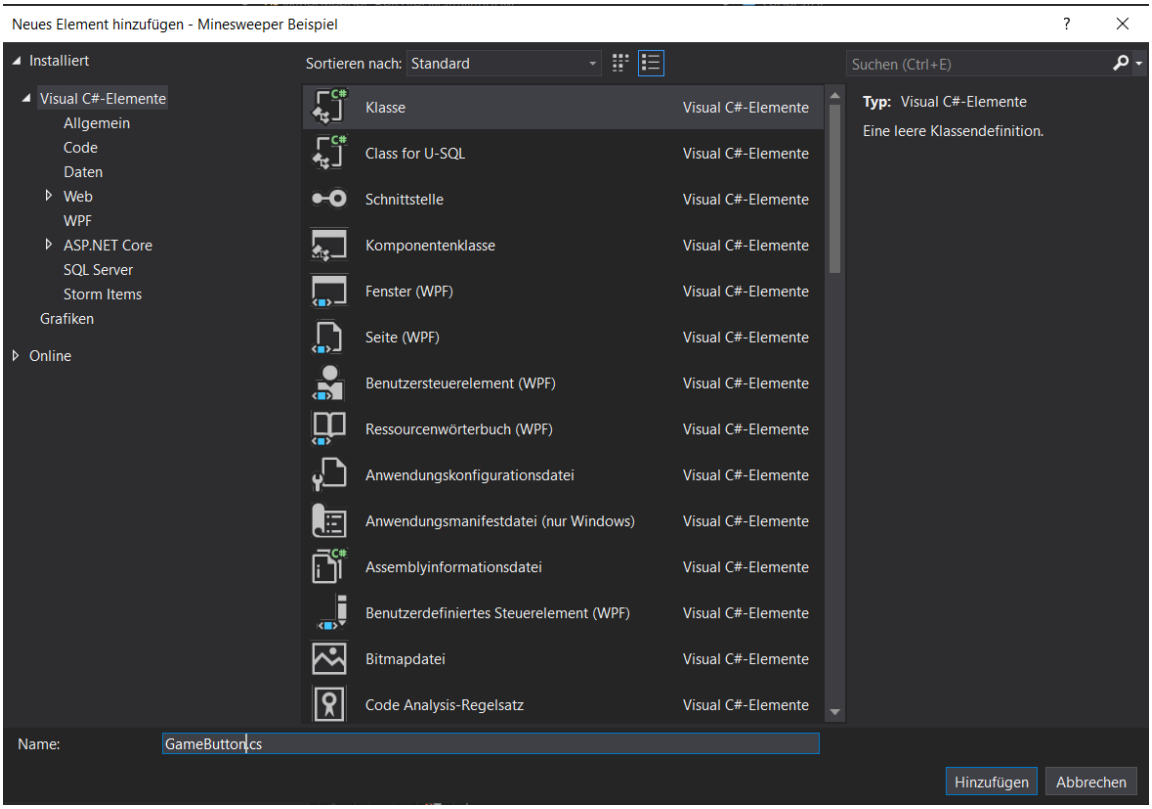
```
public MainWindow()
     {
         button_dim = 35;

         seperator = 0;

         bomb_count = 40;
         _currentBombCount = bomb_count;

         column_count = 16;
         row_count = 16;
         _fields = new GameButton[column_count, row_count];

         _bombCountTB = new TextBlock();
         InitializeComponent();
         StartNewGame();
     }
```

Klasse "GameButtons" erstellen und enum "ButtonState" und "RightClickState" hinzufügen

Hierzu müssen wir zunächst eine neue Klasse hinzufügen. Dies machen wir, indem wir im Projektmappen-Explorer mit Rechtsklick auf den Projektnamen klicken und unter Hinzufügen Klasse auswählen



Dadurch sollte sich ein Fenster öffnen, in dem wir unten den Namen der Klasse festlegen können. Hier muss der Name der Klasse von Class1.cs auf GameButton.cs geändert werden.



Nun erzeugen wir eine weitere neue Klasse die wir ButtonState nennen und ersetzen class mit enum und definiere das enum ButtonState mit den Literale Free, Mine, Number

Enum ButtonState

```
namespace Minesweeper_Beispiel
{
    public enum ButtonState : byte
    {
        Free = 0,
        Mine = 1,
        Number = 2,
    }
}
```

Außerdem erzeugen wir noch ein weiteres enum names RightClickState, welches mit den Literalen Nothing und Flag definiert wird.

Enum RightClickState

```
namespace Minesweeper_Beispiel
{
    public enum RightClickState: byte
    {
        Nothing = 0 , Flag = 1
    }
}
```

Somit haben wir die neue Klasse GameButton, die wir nun implementieren müssen.

Zunächst einmal müssen wir der Klasse Variablen zuweisen und von der Klasse Button erben lassen.

Variablen in GameButton

```
using System.Windows.Controls;
using System.Windows.Input;

        public class Gamebutton : Button
        {
                public ButtonState State { get; set; } = ButtonState.Free;
                public RightClickState RightClickState { get; private set; } = RightClickState.Nothing;
        public List<GameButton> Neighbors = new List<GameButton>();
```

Wir benötigen:

- eine Variable die den ButtonState festlegt
- eine Variable die den RightClickState festlegt
- eine Liste der Klasse GameButton, die die Nachbarn speichert

als nächstes brauchen wir den Konstruktor der Klasse GameButton der die Breite und die Höhe festlegt und dem GameButton eine Funktion hinzufügt wenn er mit der rechten Maustaste betätigt wird.

Konstruktor

```
public GameButton(int width, int height)
        {
            Width = width;
            Height = height;
                        MouseRightButtonUp += HandleRightClick;
        }
```

abschließend benötigen wir noch die Methoden SetNumberState() und FillNeighbors(List<GameButton> buttons) und HandleRightClick(object sender, MouseButtonEventArgs e)

Methoden

```
public void SetNumberState()
        {
            if (Neighbors.Any(x=>x.State == ButtonState.Mine) && State == ButtonState.Free)
            {
                State = ButtonState.Number;
            }
        }

public void FillNeighbors(List<GameButton> buttons)
        {
            Neighbors.AddRange(buttons);
        }

public void HandleRightClick(object sender, MouseButtonEventArgs e)
        {
            switch (RightClickState)
            {
                case RightClickState.Nothing:
                    RightClickState = RightClickState.Flag;
                    Content = "";
                    break;
                case RightClickState.Flag:
                    RightClickState = RightClickState.Nothing;
                    Content = "";
                    break;
            }
        }
```

Die Klasse GameButton ist somit fertig.

Methoden in MainWindow.xaml.cs

Nun kehren wir zurück in die Klasse MainWindow.xaml.cs und fügen hier die Logik des Spiels ein. Hierfür benötigen wir die Methoden :

- StartNewGame(), die das Spiel startet
- InitializeButtons(), die das GameButton Feld erstellt
- InitializeBombCount(), die den BombCount erstellt
- SetBombs(), die die Bomben verteilt
- ResetBombs(), die die Bomben neu verteilt
- HandleButtonClick(object sender, RoutedEventArgs e), die ein beim Leftclick ausgeführt wird
- Field_RightClick(object sender, MouseButtonEventArgs e), die beim Rightclick ausgeführt wird
- InitializeNeighbors(), die die Nachbarn eines Buttons festlegt
- IsOnField(Point point), die Überprüft ob ein Punkt auf dem Spielfeld liegt
- UpdateButton(GameButton sender, bool doRecursion = true), die die Buttons aktualisiert
- ManageGameOver(GameButton lastClickedButton), die überprüft ob das Spiel vorbei ist

StartNewGame()

Diese Methode löscht alle Objekte vom Canvas, legt _currentBombCount fest und initialisiert das Spielfeld

StartNewGame()

```
        private void StartNewGame()
        {
            canvas.Children.Clear();
            _currentBombCount = bomb_count;
            InitializeButtons();
            InitializeBombCount();
            ResetBombs();
        }
```

InitializeButtons()

Diese Methode füllt das GameButton array auf, legt fest welche Methode bei einem Links- bzw Rechtsklick aufgerufen wird und fügt die GameButtons dem Canvas hinzu am Ende wird InitializeNeighbors aufgerufen.

InitializeButtons()

```
private void InitializeButtons()
        {
            for (int i = 0; i < column_count; i++)
            {
                for (int j = 0; j < row_count; j++)
                {
                    _fields[i, j] = new GameButton(button_dim, button_dim);
                    _fields[i, j].Click += HandleButtonClick;
                    _fields[i, j].MouseRightButtonUp += Field_RightClick;
                    Canvas.SetLeft(_fields[i, j], i * (button_dim + seperator));
                    Canvas.SetTop(_fields[i, j], j * (button_dim + seperator));
                    canvas.Children.Add(_fields[i, j]);
                }
            }
            InitializeNeighbors();
        }
```

InitializeBombCount()

Initialisiert den BombCount und fügt ihn dem Canvas hinzu

InitializeBombCount()

```
private void InitializeBombCount()
        {
            Canvas.SetLeft(_bombCountTB, 560);
            Canvas.SetTop(_bombCountTB, 100);
            _bombCountTB.Text = $"Current Bombs: {_currentBombCount}";
            _bombCountTB.FontSize = 35;
            canvas.Children.Add(_bombCountTB);
        }
```

SetBombs()

Plaziert die Bomben zufällig auf dem Spielfeld.

SetBombs()

```
private void SetBombs()
        {
            int tempBombCount = bomb_count, randX = _random.Next(column_count), randY = _random.Next
(row_count);
            while (tempBombCount > 0)
            {
                if (_fields[randX, randY].State == ButtonState.Free)
                {
                    _fields[randX, randY].State = ButtonState.Mine;
                    tempBombCount--;
                }
                randX = _random.Next(column_count);
                randY = _random.Next(row_count);
            }
        }
```

ResetBombs()

Setzt den ButtonState von jedem GameButton auf Free, ruft SetBombs() auf und ruft für jeden GameButton SetNumberState() auf.

ResetBombs()

```
private void ResetBombs()
        {
            foreach (GameButton field in _fields)
            {
                field.State = ButtonState.Free;
            }
            SetBombs();
            foreach (GameButton field in _fields)
            {
                field.SetNumberState();
            }
        }
```

HandleButtonClick(object sender, RoutedEventArgs e)

Wenn der RightClickState des senderButtons Nothing ist setzt die Methode IsEnabled des senderButtons auf false und ruft die Methoden UpdateButton(senderButton) und ManageGameOver(senderButton) auf.

HandleButtonClick(object sender, RoutedEventArgs e)

```
        public void HandleButtonClick(object sender, RoutedEventArgs e)
        {
            GameButton senderButton = (GameButton)sender;
            if(senderButton.RightClickState != RightClickState.Nothing)
            {
                return;
            }
            senderButton.IsEnabled = false;
            UpdateButton(senderButton);
            ManageGameOver(senderButton);
        }
```

Field_RightClick(object sender, MouseButtonEventArgs e)

Ändert _currnetBombCount in Abhängigkeit der GameButtons mit dem RightClickState "Flag" und aktualisiert den Text der TextBox.

Außerdem muss die using-Direktive System.Windows.Input importiert werden

Field_RightClick

```
//Ganz oben im Programm:
using System.Windows.Input;

//in der Klasse MainWindow:
private void Field_RightClick(object sender, MouseButtonEventArgs e)
        {
            _currentBombCount = bomb_count - _fields.Cast<GameButton>().Count(button => button.
RightClickState == RightClickState.Flag);
            _bombCountTB.Text = $"Current Bombs: {_currentBombCount}";
        }
```

InitializeNeighbors()

legt für jeden GameButton fest, welche seine benachbarten GameButtons sind.

InitializeNeighbors()

```
private void InitializeNeighbors()
        {
            for (int i = 0; i < column_count; i++)
            {
                for (int j = 0; j < row_count; j++)
                {
                    List<GameButton> neighbors = (from p in _points
                                                  where IsOnField(new Point(p.X + i, p.Y + j))
                                                  select _fields[i + (int)p.X, j + (int)p.Y]).ToList();

                    _fields[i, j].FillNeighbors(neighbors);
                }
            }
        }
```

IsOnField(Point point)

Überprüft ob sich ein Punkt auf dem Spielfeld befindet.

IsOnMap(Point point)

```
private bool IsOnField(Point point)
        {
            return point.X >= 0 && point.X < column_count && point.Y >= 0 && point.Y < row_count;
        }
```

UpdateButton(GameButton sender, bool doRecursion = true)

Updatet den zuletzt gedrückten Knopf und ändert seinen Inhalt abhängig des ButtonState des Buttons.

UpdateButton(GameButton sender, bool doRecursion = true)

```
private void UpdateButton(GameButton sender, bool doRecursion = true)
        {
            switch (sender.State)
            {
                case ButtonState.Free:
                    {
                        if (!doRecursion)
                        {
                            return;
                        }
                        foreach (GameButton button in sender.Neighbors.Where(button => button.IsEnabled))
                        {
                            HandleButtonClick(button, null);
                        }
                        break;
                    }
                case ButtonState.Mine:
                    {
                        sender.Content = "X";
                        break;
                    }
                case ButtonState.Number:
                    {
                        sender.Content = "" + sender.Neighbors.Count(x => x.State == ButtonState.Mine);
                        break;
                    }

            }
        }
```

ManageGameOver(GameButton lastClickedButton)

Überprüft ob der zuletzt geklickte Button eine Miene war oder ob alle nicht Mienen Buttons gedrückt wurden.

ManageGameOver(GameButton lastClickedButton

```
private void ManageGameOver(GameButton lastClickedButton)
        {
            if (_fields.Cast<GameButton>().Count(field => field.IsEnabled) == bomb_count &&
lastClickedButton.State != ButtonState.Mine)
            {
                MessageBox.Show("Du hosd gwunga!", "Minesweeper");
            }
            else if (lastClickedButton.State == ButtonState.Mine)
            {
                MessageBox.Show("Du hosd voloan", "Minesweepre");
                foreach (GameButton field in _fields)
                {
                    UpdateButton(field, false);
                }

            }
        }
```

Zum Weiterarbeiten

Zum Abschluss noch ein paar Ideen wie Minesweeper noch weiter verbessert werden könnte:

- Möglichkeit zum Neustarten des Spiels
- Schönere Benutzeroberfläche im Spiel
- verschiedene Schwierigkeitsgrage
- In Game Timer um zu sehen wie lange man schon spielt