# Tätigkeitsberichte

## 1.10.2014

Einführung in die Materie und erstellen der GitHub-Accounts bzw. des Projektes UrPokémon. Erstellen und Weiterführung des Pflichtenhefts. Ausbau um den Ablauf der Anwendung UrPokémon.

## 8.10.2014

Fortsetzen und fertigstellen des Pflichtenhefts. Erweitern um grafische Elemente (spätere ingame-Bilder). Upload durch die grafische Oberfläche. Erstellen des UML-Diagramms (Klassendiagramm) und ausarbeiten der Funktionalitäten der einzelnen Klassen.

Abbildung 1 Rahmen

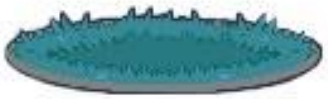

Abbildung 2 Untergrund

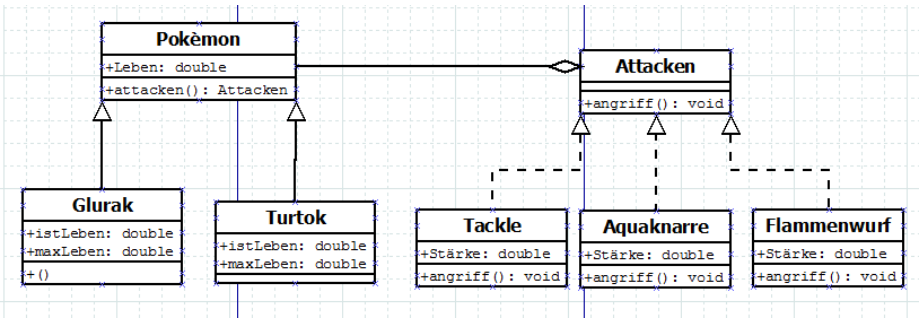


Abbildung 3 UML-Diagramm

## 15.10.2014

Erweitern des UML-Dokuments durch Kann-Kriterien und möglichen Änderungen der Funktionalitäten. Beginn der Erstellung eines Prototyps des UrPokémon-Spiels via Visual Studio. Erstellen des Klassengerüsts für eine Kommandozeilenapplikation, um die Klassen auf ihre Richtigkeit zu testen und um einen funktionierenden Prototypen erstellen zu können (da das Arbeiten mit der grafischen Oberfläche wesentlich komplexer wäre).

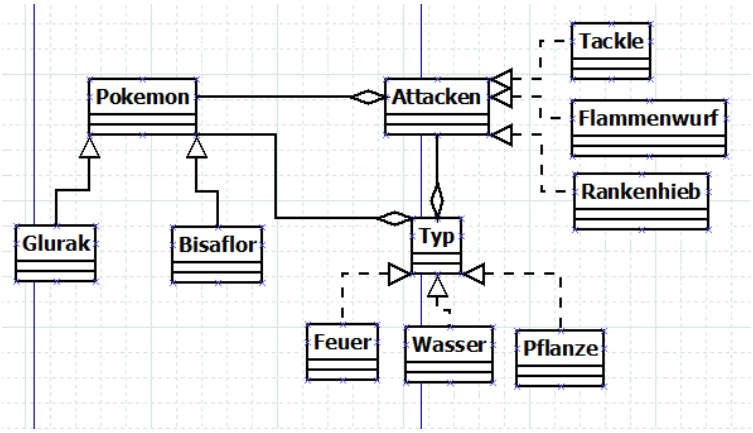


Abbildung 4 Erweitertes UML-Diagramm

## 22.10.2014

Weiterprogrammieren des Prototyps. Ausarbeiten der einzelnen Klassen (was soll in den Attacken passieren, wie sollen sich die Pokémon verhalten). Umstrukturieren der GitHub Struktur. Recherche über Rechtssituation des Bildmaterials von Nintendo.

Für den schulischen Gebrauch sind die Bildmaterialien zulässig, solange damit im späteren Verlauf keinen Absatz mit diesen gemacht wird.

## 22.10.2014-29.10.2014

Erstellen eines ersten Cmd-Prototyps des Pokémon-Spiels aufgrund des zuvor erstellten Klassendiagramms unter mehreren Einheiten in der restlichen Woche und auch am Wochenende weitergeführte Arbeiten von Fabian Feichter. Erstellen eines ersten Versuches zur grafischen Oberfläche. Recherche über das dynamische Verändern der einzelnen Bildabschnitte der grafischen Oberfläche. Implementieren von Gif-Bibliotheken, um deren Bewegung darstellen zu können. Ausbauversuch des Cmd-Modells durch die Erweiterung „Heilende Attacken“.

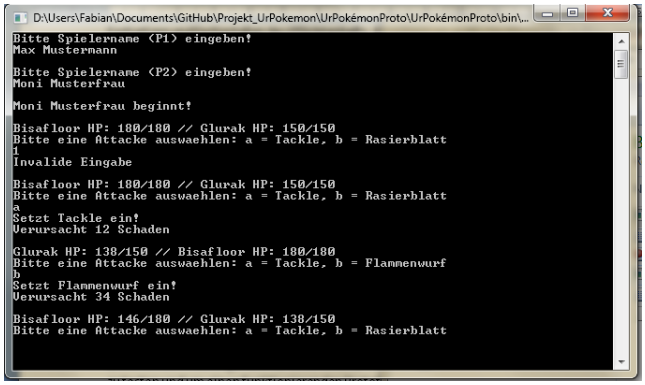


Abbildung 5 Prototyp auf CMD-Basis

/PF.NormalAttack

```
class HealingEffects : ITFAttaken
{
    static Random Rnd = new Random();
    double acc = Rnd.Next(0, 100);

    double schwank = Rnd.Next(-10, +10);
    double statusdiff = 0;

    public void angriff(double stärke, string name, double accuracy)
    {
        if (acc <= accuracy)
        {
            statusdiff = stärke + schwank;

            if (PokémonA.zug == true)
            {
               // T1.Text("Setzt " + name + " ein!");

                if (PokémonA.istLeben + statusdiff < PokémonA.maxLeben)
                {
                    PokémonA.istLeben = PokémonA.istLeben + statusdiff;
                }
                else
                {
                    PokémonA.istLeben = PokémonA.maxLeben;
                }

               // T1.Text("Heilt um " + statusdiff + " HP");
            }
            else if (PokémonB.zug == true)
            {
                //T1.Text("Setzt " + name + " ein!");

                if (PokémonB.istLeben + statusdiff < PokémonB.maxLeben)
                {
                    PokémonB.istLeben = PokémonB.istLeben + statusdiff;
                }
                else
                {
                    PokémonB.istLeben = PokémonB.maxLeben;
                }

               // T1.Text("Heilt um " + statusdiff + " HP");
            }
        }
        else
        {
            //T1.Text("Der Effekt hat keine Wirkung! ");
        }
```

Abbildung 6 Grundstruktur Heilende-Attacken

## 05.11.2014

Recherchieren von gewissen grafischen Elementen wie z.B. dynamische Textboxen. Einbinden des Programmcodes des cmd-Beispiels in das WPF-Beispiel. Erste Versuche die Ausgabe über die Textblöcke ausgeben zu lassen.

## 12.011.2014

Da es zu Fehlern mit den recherchierten Lösungen kam bzw. die Lösungsansätze nicht für unsere Bedürfnisse passend waren wurde erneut über verschiedene Button- und Textboxfunktionen recherchiert. Es wurde weitere Ansätze gefunden, welche in der folgenden Einheit implementiert werden sollen.

## 19.11.2014

Einbinden der recherchierten Lösungsansätze für das Button-Problem (Buttons waren nicht dynamisch ansprechbar). Erstmaliges Testen der Buttons und Textboxen in einem kleinen Testprogramm im Pokémon-Beispiel.


Abbildung 7 Ansprechen der Buttons

```
public void A1_Click_1(object sender, RoutedEventArgs e)
{
    A1.Visibility = Visibility.Hidden;
    A2.Visibility = Visibility.Hidden;
    A3.Visibility = Visibility.Hidden;

    B1.Visibility = Visibility.Visible;
    B2.Visibility = Visibility.Visible;
    B3.Visibility = Visibility.Visible;
}
```

Abbildung 8 Einbinden der Buttons

## 26.11.2014

Da das ansprechen der Elemente im Hauptprogramm nicht in dieser Art möglich ist, bzw. die TextBlöcke nicht in den Klassen ansprechbar sind, wird das Programm komplett umstrukturiert, um die benötigten Anforderungen zu erfüllen. Die bool'sche Variable, die für den Zug des einzelnen Spielers nötig ist, wird unnötig, da das Klicken eines Attacken-Buttons diese Aufgabe übernimmt. Die einzelnen Methoden, die Textausgaben des Schadens und dergleichen ausgaben müssen so umgebaut werden, dass sie die entsprechenden Werte zurückgeben, um sie dann mittels Textausgabe im MainWindow aufrufen zu können.

## 3.12.2014

Umstrukturieren der Buttons; einfügen eines „weiter“ Buttons, der die entsprechenden Attacken-Buttons auf Visible setzt. Dies dient zur Ermöglichung einer Textausgabe in jenem Feld in dem vorher die Buttons standen. Behebung kleiner Fehler bei der anfänglichen Ausgabe des Programms und der Schadensberechnung. Anzeigeänderungen (anzeige der Attackenstärke und –genauigkeit bei Hover des jeweiligen Buttons)

## 10.12.2014

Durch die Weiterarbeit an den letzten zwei Wochenenden von Fabian Feichter konnte der erste funktionierende Prototyp der WPF-Version fertiggestellt werden. In dieser Version ist der Ablauf nochmals vereinfacht, um ihn später jederzeit erweitern zu können. Das Programm wurde nochmals umstrukturiert um die Realisierung mittels WPF weiterführen zu können. Zusätzlich wurde noch eine Anzeige erstellt, die mittels Lebensbalken das Leben bzw. den Schaden grafisch darstellt (je mehr Leben verloren wurde, desto kürzer wird der grüne, bisher statische Balken).


Abbildung 9 Hover-Eigenschaften der Buttons


Abbildung 10 Dynamische Lebensanzeige

In der momentanen Version gibt es noch zwei Bugs, die es auszubessern gilt: Der erste Zug der Pokémon erzeugt keinen Schaden, auch wenn der Angriff an sich trifft; Die Schadensberechnung stimmt nicht mit dem errechneten Schaden des Programms überein.


Abbildung 11 Fehlerhafter Schaden beim ersten Zug


Abbildung 12 Falsche Schadensberechnung (150-21 != 85)

Ziel ist es, diese Bugs bis Weihnachten auszumerzen um den ersten fehlerfreien Prototypen vorstellen zu können. Anschließend kann dann mit der Eingabe von Spielernamen, und der dynamischen Weitergabe des Pokémon-Namens begonnen werden, da diese momentan im Hauptprogramm deklariert werden, und nicht weitergereicht werden (wie z.B. der Schaden).

## 17.12.2014

Es wurde zusätzlich ein Willkommensbildschirm hinzuprogrammiert, der die Funktionsweise der Applikation etwas erklärt. Der Fehler der Schadensberechnung beim ersten Zug wurde auch behoben.

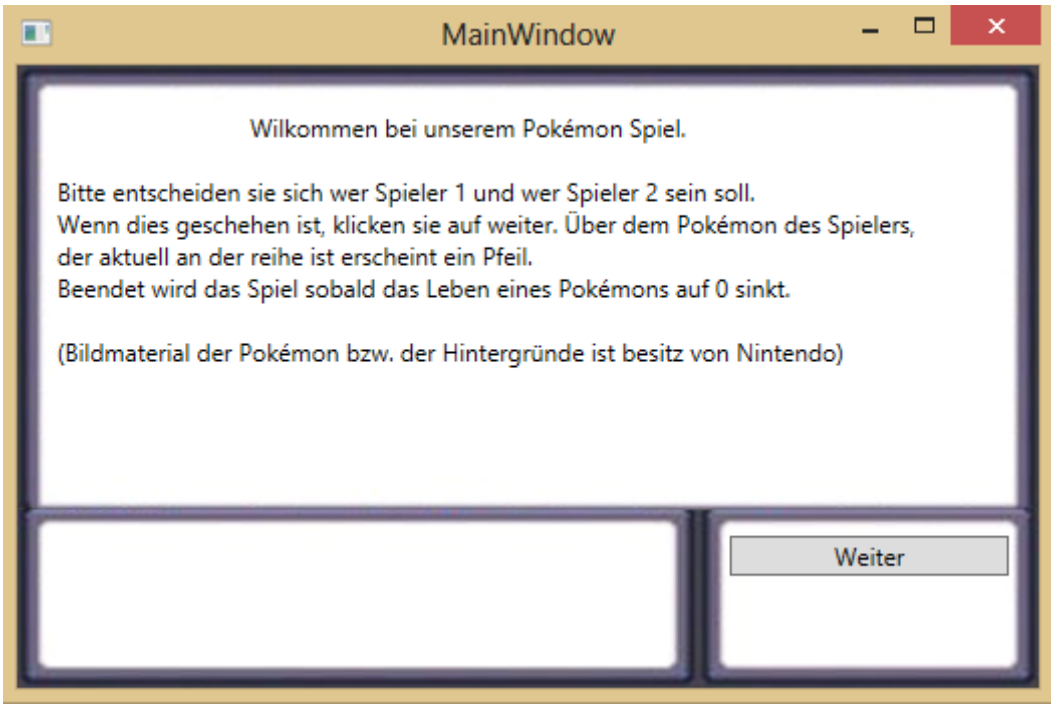


Abbildung 11: Anzeige des Willkommensbildschirm bei starten des Programms


Abbildung 12 Fehlerfreie Schadensberechnung bei 1. Zug

Als nächstes wird versucht eine Startanimation hinzuzufügen bevor die Pokémon auf dem Bildschirm erscheinen. (Hereinrollender/-springender Pokéball) Weiters wird immer noch versucht die fehlerhafte Schadensberechnung zu beheben. (Tackle macht 35 schaden, obwohl diese Attacke maximal 20 Schaden zufügen kann)

## 7.1.2015

Fehlerbehebung bei der Schadensberechnung mittels Debugger. Lokalisation der Fehlerstelle bei der Schadensberechnung der einzelnen Pokémon (Werte wie der Schaden bez. Des ist-Lebens werden vom falschen Pokémon übernommen). Erste Lösungsansätze bezüglich dieser Logikfehler werden angestellt.

## 14.1.2015

Initialisierung eines Endbildschirms der die Möglichkeit eines neuen Spieles oder des Beendens bietet.


Abbildung 13 Endbildschirm

Weiters wird recherchiert, wie eine vorher ausgeführte Attacke die folgende beeinflussen kann. Zusätzlich wurde noch ein Button Aufgeben hinzugefügt.

## 28.01.2015

Beseitigen weiterer Bugs, welche nicht immer in der RunTime aufgetreten sind, sondern eher zufälligerweise auftauchen, da ein unbekannter Zustand angenommen wird. Weiteres beseitigen von eventuell kritischen Codes, welcher solche Zustände verursacht. Beseitigen eines Logikfehlers, bei dem die Länge der Lebensanzeige einen negativen Wert annehmen könnte. Recherchieren über Sound und Anfangsanimationen und erste Versuche diese im Programm zu integrieren.

Zusätzlich wurde von Fabian Feichter noch ein weiterer Attackentyp entwickelt, welcher den nachfolgenden Zug beeinflusst. Bisafloor besitzt nun eine Attacke Focus, welches es ermöglicht, eine Attacke für den nächsten Zug aufzuladen. Es ist dadurch in der Lage, mit etwas Glück einen weitaus stärkeren Angriff zu tätigen.
Glurak auf der anderen Seite ha nun die Attacke Brüller gelernt, welches es ermöglicht, den Gegner einzuschüchtern und somit die nächste Attacke des Gegners zu schwächen, um damit einen entscheidenden Vorteil im Match zu erlangen.

# 04.02.2015

Implementieren von Sounddateien, welche bei Aktionen ausgeführt werden (z.B. bei Attacken, oder auch wenn die Lebensanzeige eines Pokémon auf null sinkt). Diese Implementierung wurde auf drei Arten getestet:

```
//1
    Uri uri = new Uri(@"D:\Users\Fabian\Documents\Schule 5BHWII 2014-15\PPM\Köllö\sounds\flamethrow (online-audio-converter.com).wav");
    var player = new MediaPlayer();
    player.Open(uri);
    player.Play();

//2
    SoundPlayer simpleSound = new SoundPlayer(@"D:\Users\Fabian\Documents\Schule 5BHWII 2014-15\PPM\Köllö\sounds\flamethrow (online-audio
    try
    {
        simpleSound.Play();
    }
    catch
    {
    }

//3
    System.Media.SoundPlayer startSoundPlayer = new System.Media.SoundPlayer(@"D:\Users\Fabian\Documents\Schule 5BHWII 2014-15\PPM\Köllö\
```

**Abbildung 14 Implementierung Sounddatei**

Die erste Variante war für unser Projekt ungeeignet, da sie nur die Sounddatei abspielt, wenn diese noch nie abgespielt wurde. Diese Variante eignet sich also eher für Hintergrundmusik/ eventuell einer Endlosschleife, damit Hintergrundmusik später noch eingebaut werden kann. Zusätzlich ist ein kleines Delay beim Aufrufen, was bei schnellen Zugwechsel nicht von Vorteil wäre.

Die zweite Variante eignet sich sehr gut für unser Projekt, da der Sound beim Ausführen der Attacke jedes Mal den Sound abspielt, und dies auch ohne Verzögerung tut. Der Programmcode wird zusätzlich durch eine Exception gesichert, da kritischer Code ausgeführt wird (die Datei könnte wo anders liegen, nicht mehr vorhanden sein etc.).

Die dritte Variante eignet sich eher schlecht, genauso wie Variante eins, da nicht immer ein Sound ausgegeben wird.