

# 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

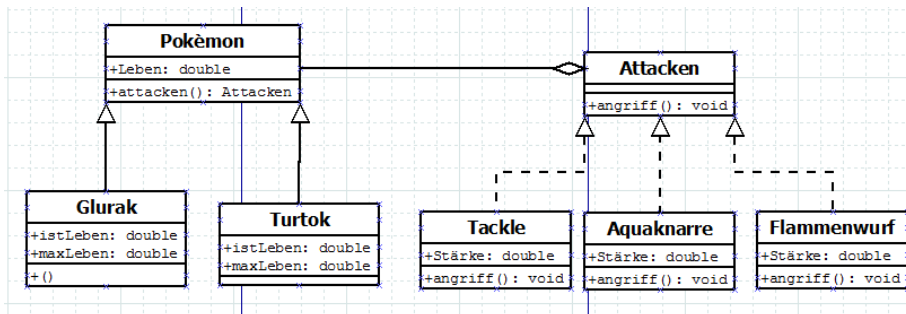


Abbildung 3 UML-Diagramm

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).

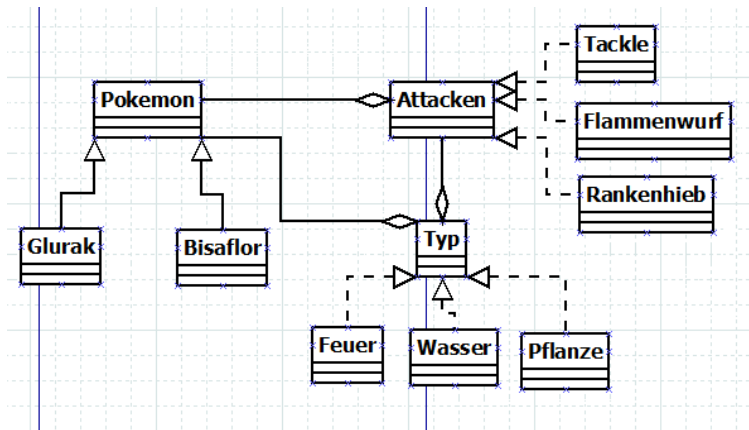


Abbildung 4 Erweitertes UML-Diagramm

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“.

```
Bitte Spielernamen <P1> eingeben!
Max Mustermann
Bitte Spielernamen <P2> eingeben!
Moni Musterfrau
Moni Musterfrau beginnt!
Bisafloor HP: 180/180 // Glurak HP: 150/150
Bitte eine Attacke auswählen: a = Tackle, b = Rasierblatt
1
Invalide Eingabe
Bisafloor HP: 180/180 // Glurak HP: 150/150
Bitte eine Attacke auswählen: a = Tackle, b = Rasierblatt
a
Setzt Tackle ein!
Verursacht 12 Schaden
Glurak HP: 138/150 // Bisafloor HP: 180/180
Bitte eine Attacke auswählen: a = Tackle, b = Flammwurf
b
Setzt Flammwurf ein!
Verursacht 34 Schaden
Bisafloor HP: 146/180 // Glurak HP: 138/150
Bitte eine Attacke auswählen: a = Tackle, b = Rasierblatt
```

Abbildung 5 Prototyp auf CMD-Basis

```
/PF.NormalAttack
class HealingEffects : ITFAttacken
{
    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 ausgeben 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 ( $150 - 21 \neq 85$ )

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).