|  |
| --- |
|  |
| Pokémon World Online |
|  |
| C:\Users\Niels\Desktop\logo.jpgEin Klassiker als MMORPG |

**Besonderelernleistung**

28 Oktober 2012

Verfasst von: Niels Becker

Inhaltsverzeichnis

[1. Einleitung 1](#_Toc349914403)

[2. Allgemeines 2](#_Toc349914404)

[2.1 Pokémon 2](#_Toc349914405)

[2.1.1 Konzept 2](#_Toc349914406)

[2.1.2 Editionen 3](#_Toc349914407)

[2.2 GDI+ 4](#_Toc349914408)

[3. Vorgehensweise 5](#_Toc349914409)

[3.1 Aufgabenstellung 5](#_Toc349914410)

[3.2 Umsetzung 5](#_Toc349914411)

[4. Komponenten 6](#_Toc349914412)

[4.1 Client 6](#_Toc349914413)

[4.2 Server 7](#_Toc349914414)

[4.3 Datenbank 8](#_Toc349914415)

[4.4 Karten-Editor 8](#_Toc349914416)

[5. Grafik-Engine 9](#_Toc349914417)

[5.1 Aufbau 9](#_Toc349914418)

[5.1.1 Grafische Ausgabe 9](#_Toc349914419)

[5.1.2 Eingabe 9](#_Toc349914420)

[5.1.3 Ressourcen Verwaltung 10](#_Toc349914421)

[5.1.4 Sound 10](#_Toc349914422)

[5.2 GUI-System 10](#_Toc349914423)

[5.3 Eventsystem 11](#_Toc349914424)

[5.2.1 Event-Delegat 11](#_Toc349914425)

[5.2.2 Event-Handler 11](#_Toc349914426)

[5.2.3 Realisierung 11](#_Toc349914427)

[6. Server-Client Komunikation 12](#_Toc349914428)

[6.1 Verschlüsselung 13](#_Toc349914429)

[7. Map-Management 14](#_Toc349914430)

[7.1 Wegdaten 14](#_Toc349914431)

[7.2 Script-Phraser 15](#_Toc349914432)

[8. Ergebnis 16](#_Toc349914433)

[8.1 Übersicht über das Spiel 16](#_Toc349914434)

[8.2 Selbstkritik 16](#_Toc349914435)

[8.3 Mögliche Erweiterungen 16](#_Toc349914436)

[9. Anhang 17](#_Toc349914437)

[9.1 Quellen 17](#_Toc349914438)

[9.2 Anleitung zum Spielen 17](#_Toc349914439)

[9.2.1 Tastaturbelegung 17](#_Toc349914440)

[9.2.2 Konsolenbefehle 17](#_Toc349914441)

[9.3 Diagramme 17](#_Toc349914442)

[9.3.1 Client Klassendiagram (Game Abschnitt) 18](#_Toc349914443)

[9.3.2 Client Klassendiagram (Engine Abschnitt) 19](#_Toc349914444)

[9.3.3 Server Klassendiagram 20](#_Toc349914445)

[9.3.4 Datenbank ER-Modell 20](#_Toc349914446)

[9.3.5 Map-Editor Klassendiagram 20](#_Toc349914447)

[9.4 Code Ausschnitte 21](#_Toc349914448)

[9.4.1 Map-Einstellung Beschreibung 21](#_Toc349914449)

Pokémon World Online

Ein Klassiker als MMORPG

# 1. Einleitung

Die Projektidee entstand im Frühling 2012 während ein paar Freunde und ich Pokémon gespielt haben. Uns ist aufgefallen das es keine Möglichkeit gibt gemeinsam zu spielen. Daraufhin überlegten ich mir wie ein Multiplayer[[1]](#footnote-1) Pokémon Spiel aussehen könnte.

Bei der Suche im Internet nach einer Multiplayer Version von Pokémon wurde ich leider nicht fündig. Also beschloss ich, dass Projekt in Angriff zu nehmen, ohne dabei an eine besondere Lernleistung zu denken.

Zu dem Titel kam ich, da ich mir überlegt, dass man auch die normale Story[[2]](#footnote-2) des Spiels gemeinsam Spielen sollen kann und man bei dieser in einer Welt frei herumlaufen kann. In der weitern Ausarbeitung werde ich deshalb mein Spiel mit PWO abkürzen.

Im Laufe der Programmierung bemerkte ich, dass das Projekt größer wird als erwartet und da ich mir sicher war eine besondere Lernleistung im Bereich Datenverarbeitung zu erbringen, entschied ich mich für dieses Projekt.

# 2. Allgemeines

## 2.1 Pokémon

Pokémon (jap. ポケットモンスター, dt. Taschen-Monster) sind erfundene Wesen, die meist auf Tieren oder Gegenständen basieren. Erfunden und ursprünglich herausgebracht von Satoshi Tajiri und der von ihm gegründeten japanischen Softwarefirma GAME FREAK Inc. stellt Pokémon neben Super Mario mittlerweile eine der wichtigsten Marken für Nintendo dar. 1996 wurde in Japan das erste Pokémon-Spiel herausgebracht, dem eine Anime-Serie, ein Sammelkartenspiel, bislang 15 Kinofilme, eine Menge von Manga und weitere Merchandising-Produkte folgten. Bis heute wurden über 200 Millionen Pokémon-Spiele verkauft.

### 2.1.1 Konzept

Die Grundidee von Pokémon basiert auf dem Sammeln von Tieren, wie verschiedenen Insekten oder Kaulquappen, was Satoshi Tajiri in seiner Kindheit eine riesige Freude bereitete. Als er älter wurde, wollte er Kindern den selben Spaß vermitteln, den er selbst hatte, und setzte darum seine Idee Kreaturen zu fangen in die Tat um. Der Spieler schlüpft am Anfang beinahe jeden Spiels in die Rolle eines Pokémon-Trainers, der vor allem zwei wichtige Aufgaben im Spielverlauf zu erfüllen hat. Zum einen gilt es den Pokédex, der die Daten aller gesehenen und gefangenen Pokémon speichert, zu vervollständigen. Das zweite Ziel besteht darin, möglichst starke Pokémon zu fangen und diese zu trainieren, sodass man mit Hilfe dieses Teams die Pokémon-Liga bezwingt und somit der neue Champ der Region wird. Nicht nur in den Spielen werden diese Ziele verfolgt, auch im Anime und Manga, sowie im Sammelkartenspiel liegt der Schwerpunkt im Erreichen dieser.

Das Fangen von Pokémon wird durch Einsatz eines speziellen und hoch entwickelten Gerätes, dem Pokéball, ermöglicht. Pokébälle gibt es in vielen verschiedenen Ausführungen, wobei jeder von ihnen andere Stärken und Schwächen aufweist. Soll ein Pokémon gefangen werden, wird ein Pokéball auf das zu fangende Pokémon geworfen, das in einem Energiefeld in den Ball eingesaugt wird. Mit etwas Glück bleibt das meist durch einen Kampf geschwächte Pokémon im Pokéball und gehört ab diesem Zeitpunkt dem Trainer. Hat der Trainer jedoch nicht ausreichend Erfahrung im Umgang mit Pokémon, kann es sein, dass das neu gefangene Pokémon ihm noch nicht gehorcht. Es können lediglich wilde Pokémon gefangen werden; das „Krallen“ von Pokémon anderer Trainer funktioniert nur in den beiden für den Gamecube erschienenen Spielen Pokémon Colosseum und Pokémon XD.

Es gibt bislang 649 verschiedene Pokémon mit 17 verschiedenen Typen. Einige von ihnen besitzen auch Typenkombinationen, die aus zwei verschiedenen Typen bestehen. Je nach Kombination sind Pokémon auf verschiedene Typen mehr oder weniger anfällig. Diese Anfälligkeiten sind bei Kämpfen von großer Bedeutung. Ziel ist es, ein Team von bis zu sechs Pokémon zusammenzustellen, das möglichst wenig Schwächen und möglichst viele Stärken aufweist. Mit diesen Pokémon kann dann gegen spielinterne Trainer oder gegen Freunde gekämpft werden. Seit Pokémon Feuerrot und Blattgrün ist es möglich, diese Kämpfe kabellos und nicht wie bisher mit dem Linkkabel durchzuführen. Seit der vierten Spielgeneration ist es sogar möglich über das Internet, mit Hilfe der Nintendo Wi-Fi Connection zu Kämpfen und zu Tauschen. Der Tausch ist ein weiteres wichtiges Element bei Pokémon, da in jedem Spiel unterschiedliche Pokémon auftauchen. Will man beispielsweise nun eines haben, das aber in seinem Spiel nicht fangbar ist, muss man zum Tausch greifen und sich das Pokémon von einem Freund oder eben über das Internet von anderen Trainern besorgen. Man kann Pokémon auch von einer Spielgeneration in eine andere übertragen, dies geht jedoch nur in die Richtung der neueren Spielgeneration, in eine ältere Generation kann man nicht mehr zurücktauschen.

Der Großteil der Pokémon ist fähig, sich zu entwickeln. Es gibt eine Vielzahl an Möglichkeiten, wie eine Entwicklung zu Stande kommen kann. Die gängigste Methode ist der Levelaufstieg (auch „Lv.-Up“). Dabei entwickelt sich das Pokémon, sobald es ein bestimmtes Level erreicht hat. Manchmal benötigt ein Pokémon zusätzlich zum Levelaufstieg viel Zuneigung zum Trainer, oder muss eine bestimmte Attacke erlernen. Oft sind zur Entwicklung auch spezielle Items nötig, wie zum Beispiel bestimmte Steine. Eine letzte Möglichkeit der Entwicklung ist noch der Tausch. Einige Pokémon entwickeln sich nur, wenn man sie, teils sogar mit einem bestimmten Item, tauscht.

Dieser Prozess ist durch Zucht auch umkehrbar. Beinahe jedes Pokémon, ausgenommen von legendären und Baby-Pokémon, ist in der Lage, gemeinsam mit einem zweiten Pokémon ein Ei zu legen, aus dem dann die niedrigste Stufe des Pokémon schlüpft. Bei wenigen Ausnahmen muss das zu züchtende Pokémon noch ein spezielles Item tragen. Durch Zucht können viele Pokémon Attacken lernen, die sie sonst nicht lernen würden.

### 2.1.2 Editionen

* Game Boy:
  + Poketto Monsutā Midori/Aka (26.02.1996) (nur in Japan)
  + Pokémon Rote und Blaue Edition (26.02.1996)
  + Pokémon: Gelbe Edition (15.05.2000)
* Game Boy Color:
  + Pokémon Goldene und Silberne Edition (05.04.2001)
  + Pokémon Kristall-Edition (01.11.2001)
* Game Boy Advance:
  + Pokémon Rubin- und Saphir-Edition (24.05.2003)
  + Pokémon Feuerrote und Blattgrüne Edition (01.09.2004)
  + Pokémon Smaragd-Edition (15.10.2004)
* Nintendo DS:
  + Pokémon Diamant- und Perl-Edition (26.05.2007)
  + Pokémon Platin-Edition (22.04.2009)
  + Pokémon Goldene und Silberne Edition (HeartGold / SoulSilver) (25.03.2010)
  + Pokémon Schwarze und Weiße Edition (04.03.2011)
  + Pokémon Schwarze und Weiße Edition 2 (12.09.1012)

## 2.2 GDI+

## 2.3 LUA

# 3. Vorgehensweise

## 3.1 Aufgabenstellung

Zunächst soll der Spieler die selber Story, wie in der Rubin Edition des originalen Spieles, durchspielen können. Dazu soll die originale Karte verwendet werden. Ebenfalls sollen alle Pokémon und Items[[3]](#footnote-3) aus dem original übernommen werden.

Da es ein Multiplayer Spiel seien soll, sollen sich die Spieler von Anfang an sehen können und miteinander die Story des Spiels durcharbeiten können.

Ebenfalls sollen die Spieler untereinander Handeln, Reden und Kämpfen können.

Hinzukommt eine Rangliste, an der man erst nach dem Geschichtlichen-Teil, d.h. nachdem man die klassische Geschichte der Pokémon Reihe durchgespielt hat, teilnehmen kann. Hierbei sollen die Spieler untereinander ihre Stärke messen können.

## 3.2 Umsetzung

Ich entschied mich nicht für eine Grafik-Engine aus dem Internet, sondern wollte selber eine mit den Vorgaben von Windows (GDI+) entwerfen, da ich dies bis lang noch nie getan hatte.

Bei der Client-Server-Kommunikation entschied ich mich für das TCP-Protokoll und einer selbst entworfenen Verschlüsselungs-Methode.

Zu Datenverwaltung benutze ich auf der Server Seite eine MySQL-Datenbank und auf der Client Seite einfache Textdateien, die in einer bestimmten Form aufgebaut seien müssen.

Um eine einfache Bearbeitung der Quests[[4]](#footnote-4) zu gewährleisten, entschied ich mich für LUA, eine Schriftsprache mit C++ Bindung.

Die gesamte Karte setzt sich aus einzelnen Kartenabschnitten zusammen, welche jeweils einzeln geladen werden können.

Da ich bemerkte, dass die Erstellung der einzelnen Kartenabschnitte per Hand zu lange dauerte, entschied ich mich einen Karten-Editor zu programmieren. Dieser soll die Erstellung erleichtern, indem man durch einfaches zusammen klicken die Objekte und Wegdaten[[5]](#footnote-5) platzieren kann.

# 4. Komponenten

## 4.1 Client

(Siehe Klassendiagramm 9.3.1 und 9.3.2)

Im Client gibt es zwei verschiedene Bereiche: Den Engine-Teil und den Game-Teil. Ersterer ist unabhängig von dem Game-Bereich und könnte für eine andere Anwendung wieder verwendet werden. Die Engine wird in *5. Grafik-Engine* genauer erläutert. Der Game-Teil, der alle Inhalte des Spiels beinhaltet benutzt die Engine zum Rendern der Grafiken und zum Abfragen der Inputs. Dieser Teil kann nicht ohne die Engine funktionieren.

Das Spiel ist in verschiedene Phasen unterteilt s.g. GamePhases. Wenn man das Spiel startet kommt man zunächst in die Menü-Phase, in der man sich einloggen muss. Als nächstes folgt die Lade-Phase, die alle nötigen Ressourcen lädt. Darauf folgt die World-Phase in der man sich in der Welt umherbewegen kann und wenn man einen Kampf beginnt, wechselt man in die Battle-Phase. Je nachdem welche Phase aktiv ist, wird ein anderes UI[[6]](#footnote-6) angezeigt oder nur bestimmt Tastenanschläge abgefragt.

Des Weiteren befinden sich im Game-Abschnitt die Verwaltungs-Klassen für die Karte, die Spielersteuerung, die Entitys (NPCs und andere Spieler) und die Kämpfe.

Der Aufbau der Map[[7]](#footnote-7) Verwaltung wird in *7. Map-Management* genauer erläutert. Bei der grafischen Ausgabe muss man beachten, dass der Spieler sich immer nur auf einem Map Element bewegen kann. Also brauch auch nur dieses angezeigt werden um Speicher und Rechenzeit zu sparen. Allerdings muss man, wenn sich der Spieler den Rand der aktiven Map nähert das nächste angrenzende Map Element einblenden. Deswegen werden in dem Map Container die aktive Map und ihre bis zu 4 angrenzenden Elemente mit einander verknüpft und als eins angezeigt. Außerdem werden hier die Ränder erweitert, falls keine Map angrenzt, da man sonst ins Schwarze gucken würde. Dazu wird eine Grafik, die dem Rand der Map entspricht, wiederholend in den schwarzen Bereich gelegt.

Sie Spielersteuerung ist relativ einfach aufgebaut, sie fängt die Verschiedenen Tasten-Anschläge des Spielers ab und führt die dementsprechende Funktion aus. Z.B. wird bei den Bewegungstasten (w, a, s, d und Pfeiltasten) nur die Karte vom Map-Manager bewegt und ein Bewegungs-Packet an den Server gesendet.

Im Entity-Manager werden die Entitys, d.h. die NPCs und die anderen Spieler, verwaltet. Hier werden die Position und die Bewegung der Entitys berechnet und gespeichert. Wenn sich der Spieler bewegt, bzw. die Map sich bewegt, dürfen sich die Entitys nicht verschieben. Sie müssen, während die Animation für die Bewegung läuft, an derselben Map-Position bleiben. Hinzu kommt, dass die Position der Entitys in Feldern der Map gemessen wird, die Position auf dem Bildschirm allerdings in Pixeln. Von daher muss dies auch noch relativ zur Fenstergröße umgerechnet werden.

Die Kampf-Verwaltung ist zuständig für alles was in einem Kampf passiert. Sie sagt dem Server welche Attacke der Spieler ausführen möchte und spielt die dementsprechenden Animationen ab. Zusätzlich verwaltet sie die Pokémon des Spielers.

## 4.2 Server

(Siehe Klassendiagramm 9.3.3)

Der Server wird durch eine Singleton-Klasse[[8]](#footnote-8) dargestellt. Diese erstellt und verknüpft die verschiedenen Verwaltungsklassen untereinander, mit der Netzwerk-Kommunikation (siehe *6. Server-Client Kommunikation*) und der MySQL-Datenbank.

Zur einfacheren Abfrage der Daten aus der Datenbank dient die MySQL-Klasse, deren Query-Funktion ein 2D-Feld zurückgibt. Aus dem man dann einfach über die Indices die gewünschten Daten auslesen kann.

Genauso wie im Client gibt es eine Map Verwaltung, die so ähnliche funktioniert, allerdings ohne grafischen Ausgabe (siehe *7. Map-Management*).

Alle Objekte die sich auf der Map befinden, werden durch die Basisklasse Entity dargestellt. Unteranderem sind das die Spieler und die NPCs. Damit diese sich untereinander sehen muss bei jedem Schritt den sie machen überprüft werden ob ein neuer Spieler in Sichtweite gekommen ist oder ein andere sie verlassen hat. Dies geschieht in dem Player-Manager. Ebenfalls überprüft er auch die Login-Daten der Spieler und fragt alle Items, Pokémon usw. aus der Datenbank ab.

Die Items werden im Item-Manager verwaltet. Da es verschiedene Typen von Items gibt, Pokéball, Trank, Beere, usw. werden alle Items durch die Basisklasse ItemBase dargestellt. Jeder Itemtype hat eine Funktion für das Benutzen des Items auf ein Pokémon, auf den Spieler und zum Übergeben des Items an ein Pokémon. Ein Item besitzt 4 verschiedene Values, in denen die Werte des Items gespeichert werden. Je nachdem von welchem Itemtype das Item ist, haben die Werte unterschiedliche Funktionen. Bei einem Pokéball sagt die Value1 die Stärke des Pokéballs an, hingegen bei einem Trank besagt diese um wie viel KP sich das Pokémon heilt. Damit nicht jedes Mal, wenn ein Item erstellt wird, eine Datenbank Abfrage gemacht werden muss, wird am beim Starten alle Item-Prototypen in den Speicher geladen. Ein Item-Prototype ist ein Item, dass keinen Besitzer hat und unverändert ist. Aus ihm kann man dann ein Item erstellen und ihm einen Besitzer zuweisen.

Bei den Pokémon ist es ähnlich wie bei den Items. Auch diese besitzen Prototypen die vom Pokemon-Manager zu Beginn eingelesen werden. Allerdings gibt es hier verschiedene Prototypen. Die normalen Prototyen sind für die Pokémon der Spieler. Sie beinhalten die Typen, alle Statuswerte, Geschlecht, Fangchangse, usw. der Pokémon. Hinzukommen Prototypen für Wilde Pokémon und Trainer Pokémon. Die Wilden Pokémon haben noch zusätzliche Attribute für das min. und max. Level mit denen sie auftauchen können. Die Trainer Pokémon besitzen zusätzliche Attribute für das Level und die Attacken, da diese bei den Trainer Pokémon festgelegt sind. Der Pokemon-Manager verwaltet zusätzlich noch die Attacken der Pokémon. Er berechnet auch die Erfahrung, die Pokémon für das nächste Level benötigen und wie viel sie nach einem Kampf bekommen.

Die Aufgaben des Spielers werden in dem Quest-Manager verwaltet. Dieser liest beim Starten alle Quests ein, da diese in externen Dateien gespeichert sind um die Übersicht zu behalten. Die Quests sind in LUA geschrieben und jede Quest-Datei enthält eine oder mehr Funktionen, die dann einfach aufgerufen werden können. Als Übergabe-Parameter haben diese den Spieler, der die Quest ausführt und zusätzliche Informationen; z.B. bei einem Dialog mit einem NPC, die Daten des NPCs. Quests können durch verschiedene Interaktionen auf der Map, wie z.B. der Dialog mit einem NPC oder ein Trigger-Feld über das man läuft, ausgelöst werden. Aber auch andere Ereignisse, z.B. der Login, können Auslöser sein.

## 4.3 Datenbank

(Siehe ER-Modell 9.3.4)

## 4.4 Karten-Editor

(Siehe Klassendiagramm 9.3.5)

# 5. Grafik-Engine

Die Grafik-Engine ist der Grundbaustein für den Client. Sie verwaltet das Windowsfenster und alles was darin angezeigt wird. Hinzu kommt die Ausgabe der Sounds und das abfangen der Eingaben des Benutzer. Eine Besonderheit ist, dass die Grafik-Engine unabhängig von dem Spiel ist. D.h. sie könnte ohne Änderungen in einem anderem Projekt wiederverwendet werden.

## 5.1 Aufbau

(Siehe Klassendiagramm 9.3.2)

Die Engine lässt sich in vier Bereiche aufteilen: die Eingabe, die Grafische Ausgabe, die Ressourcen Verwaltung und die Sound Ausgabe. Die Hauptklasse der Engine verwaltet alle diese Bereiche. Zusätzlich sorgt sie dafür, dass alles im selben Zeitabstand passiert, d.h. dass die Framerate[[9]](#footnote-9) konstant bleibt. Dies geschieht durch einen Timer der die Zeit für den aktuellen Frame misst und dem entsprechend die Pause vor dem nächsten Frame verkürzt.

### 5.1.1 Grafische Ausgabe

Die grafische Ausgabe geschieht in dem Render-Window, welches ein leeres Fenster ist auf das gezeichnet werden kann. Der Scene-Manager besitzt ein Bild, dass der Größe des Fensters entspricht, der s.g. Backbuffer[[10]](#footnote-10). Alle Grafiken, die ausgegeben werden sollen, werden zuerst auf den Backbuffer gezeichnet. Im nächsten Frame wird der Backbuffer auf das Fenster projektiert. Dies verhindert ein Flackern des Bildschirmes.

Alle Grafik-Objekte die ausgegeben werden können müssen von der Basisklasse 'DrawAble' sein. Diese beinhaltet das zu zeichnende Bild, dessen Position und seine Größe. Diese DrawAbles müssen einem 'Layer' zugeordnet werden. Ein Layer ist eine Zeichen Ebene, die mehrere DrawAbles beinhalten kann. Zusätzlich um ein pseudo 3D-Effeck[[11]](#footnote-11) hervorzurufen haben diese Layer einen Z-Index. Objekte die weiter im Hintergrund gezeichnet werden sollen, wie z.B.: der Map Hintergrund, werden einem niedrigeren Layer zugeordnet. Andere Objekte die immer im Vordergrund seinen sollen, wie z.B. das UI, werden auf dem höchsten Layer gerendert.

Damit nicht jeden Frame alle Objekte neu gezeichnet werden müssen, wird in gespeichert ob sich die Grafik verändert hat. Dies kann sowohl ein neues Bild als auch eine Positions- oder Größenänderung sein.

### 5.1.2 Eingabe

In jedem Frame werden alle Nachrichten des Render-Windows verarbeitet. D.h. es wird abgefragt, ob z.B. eine Taste gedrückt oder die Maus bewegt wurde. Je nachdem wird das entsprechende Event ausgelöst. Die Ereignisse die durch den Benutzer ausgelöst wurden, werden durch den InputManager abgefangen. Alle anderen Ereignisse werden direkt weitergeleitet. Dazu zählen auch die OnPaint Ereignisse, die an den Scene-Manager weitergeleitet werden, damit dieser den aktuellen Frame zeichnen kann.

### 5.1.3 Ressourcen Verwaltung

Der Ressourcen-Manager verwaltet alle Dateien die während der Laufzeit geladen werden. Er sorgt dafür, dass die Dateien, die mehrfach verwendet werde, nicht jedes Mal neu geladen werden müssen. Dies führ zu einer deutlichen Verbesserung der Spiel-Performance, da z.B. die Textur eines Buttons über ein duzend mal geladen werden muss und so nur einmal eingelesen wird.

Der Ressourcen-Manager verwaltet alle Ressourcen. Es gibt allerdings verschiedene Arten von Ressourcen z.B. Bilder, Script Dateien, usw. , die alle unterschiedlich eingelesen werden müssen. Deshalb macht an dieser Stelle eine Vererbung Sinn. Die Basis Klasse 'Resource' ist eine abstrakte Klasse, welche die pur virtuellen Funktionen 'load()', zum einlesen der Datei, und 'release()', zum wieder freigeben, besitzt. Je nach Dateitype wird eine andere Ressource erstellt.

Wenn z.B. der UI-Manager ein neuen Butten erstellt, fragt er bei dem Ressourcen-Manager nach der Datei, wurde diese bereits geladen gibt er sie einfach zurück, ansonsten lädt er die Datei und speichert sie für das nächste Mal ab.

### 5.1.4 Sound

Die Sound Ausgabe basiert auf der Open-Source Bibliothek "libZPlay". Dies ist eine einfach Sound Bibliothek, die alle gängigen Formate abspielen kann. Man erstellt ein Sound-Objekt und ruft dann einfach die Play und Stop Funktionen auf. <

## 5.2 GUI-System

Das GUI-System verwaltet alle Menüs und Controls[[12]](#footnote-12) die dem Benutzer angezeigt werden. Die Controls werden in dem UI-Manager erstellt und aktualisiert. Er besitzt einen eigenen Layer, der die höchste Position besitzt, damit das UI über allen anderen Grafiken angezeigt wird.

Damit man die Größe der Controls anpassen kann, ohne das sie verzerrt werden, wird die Hintergrundgrafik des Controls dynamisch generiert. Ein Control besteht aus 9 verschiedenen Teilen: 4 Ecken, 4 Seiten und die Mitte. Die Fläche zwischen den Ecken wird am Rand mit den Seiten teilen ausgefüllt, indem diese sich immer wieder in X- bzw. Y-Richtung wiederhohlen. Das selber wird mit dem mittleren Teil gemacht, nur dass dieses sich in X- und Y-Richtung wiederholt.

Ein Control besitzt mehrere Events, z.B. MouseEnter oder MouseLeave. Je nach Controlart können noch andere Events vorhanden sein, z.B.: OnClick bei einem Button oder TextChanged bei einer TextBox.

Ebenfalls wie die Layer besitzen die Controls einen Z-Index, damit man die Reihenfolge, in der sie Dargestellt werden, bestimmen kann. Jedes Mal wenn ein neues Control erstellt wird müssen alle Controls neu sortiert werden, damit dieses auch in der richtigen Reihenfolge gerendert wird.

Ein Zusammenschluss aus mehreren zusammenhängeenden Controls wird als UI bezeichnet. Z.B. der Login Screen besteht aus dem Hintergrundbild, einem Panel, zwei Buttons, zwei Labes und zwei TextBoxen. Alle diese Controls werden in einer Layoutdatei gespeichert. Diese Datei enthält Informationen über die Controlarten, Positionen, Größen, usw.. Eine Layout ist in XML-Form aufgebaut. Mit dem 'Window'-Tag erstellt man ein neues Control. Der 'Property'-Tag bestimmt die Eigenschaften des Controls. Dabei ist zu beachten, dass die Position des Controls immer relativ zu dem übergeordneten Control berechnet wird.

## 5.3 Eventsystem

Das Eventsystem besteht grob gesehen aus zwei Kompetenten: dem Event-Handler und dem Event-Delegaten. Der Handler ist der Auslöser eines Event. Ihm können Delegaten zugewiesen werden, die eine Verknüpfung mit einer Funktion besitzen.

### 5.2.1 Event-Delegat

Ein Event-Delegat repräsentiert eine Verknüpfung zu einer Funktion, die bei einem ausgelösten Event aufgerufen wird.

Es gibt zwei verschiedene Arten von Delegaten: einen für statische Funktionen und einen für Member Funktionen. Eine statische Funktion kann ohne Probleme direkt aufgerufen werden. Bei einer Member Funktion muss das Objekt bekannt sein. Dieses muss bei der Erstellung mit übergeben werden.

Die Verknüpfte Funktion muss den Rückgabewert void besitzen, da es keinen Sinn macht diesen abzufragen. Allerdings kann die Funktion beliebige Übergabe Parameter besitzen, abhängig von dem Event-Handler mit dem sie verknüpft werden soll.

### 5.2.2 Event-Handler

Ein Event-Handler ist der Auslöser eines Events.

Auch hier gibt es zwei verschiedene Typen von Handlern: den Single-Event-Handler und den Multi-Event-Handler. Dem Namen entsprechen kann dem Single-Handler nur eine Delegat zugeordnet werden. Dieser ist deutlich schnelle als der Multi-Handler, da dieser keine Liste der Delegaten besitzt sondern direkt die Verknüpfung abrufen kann.

Ein Event-Handler kann mit beliebig vielen Argumenten erstellt werden, je nachdem was man braucht.

### 5.2.3 Realisierung

# 6. Server-Client Komunikation

Der Server kommuniziert mit dem Client über das TCP-Protokoll. Dabei werden verschiedene Pakete ausgetauscht. Damit der Empfänger auch weiß um was für ein Paket es sich handelt, besitzt jedes Paket ein Pakettype. Hinzu kommt, dass auch die Länge der Daten in dem Paket mit gesendet wird, damit der Empfänger verschiedene Pakete auseinander halten kann, falls mehrere Pakete auf einmal eintreffen.

Damit das Empfangen und Sender der Pakte nicht Blockiert ist dies jeweils in einem Thread ausgelagert. D.h. das Programm läuft parallel weiter. Der Thread zum Senden besitzt eine Liste mit den Daten die gesendet werden sollen. Bei jedem schleifendurchlauf wird diese Überprüft und falls ein Element vorhanden ist, wird dieses aus der Liste gelöscht und gesendet. Der Empfängerthread funktioniert so ähnlich, nur das hier die empfangen Pakete zu der Liste hinzugefügt werden. Diese werden anschießend im nächsten Frame des Client bzw. Updateaufruf des Servers verwertet. Im Server wird zusätzlich noch die 'select'-Funktion verwendet, da nur ein Thread für alle Clients zuständig ist und gleichzeitig, bzw. nacheinander von allen Empfangen werden soll. Wenn für jeden Client ein Thread erstellt würde, würde dies die CPU-Belastung und den Speicherverbrauch, bei steigender Clientanzahl, extrem erhöhen, da jeder Thread seinen eigenen Stack besitzt.

Alle Datentypen die über das Netzwerk übertragen werden, müssen logischer weise sowohl im Client als auch auf dem Server vorhanden sein. Dazu gibt es eine gemeinsame Headerdatei, die in beiden Projekten eingebunden wird. In dieser Datei sind z.B. die Pakettypen definiert, aber auch ganze Strukturen wie die Item- oder Pokemon-Struktur.

**Anwendungsbeispiel**:

(siehe Sequenzdiagram 9.3.6)

Bei einem Verbindungsaufbau des Clients überprüft der Server zuerst, ob noch ein Platz frei ist. Falls der Server voll ist, sendet der Server eine Fehlermeldung zurück und beendet die Verbindung. Anderenfalls sendet der Server 'OK' und überträgt den zufällig generierten Client-Schlüssel, der zur Verschlüsslung aller weiteren Pakete dient. Die hergestellte Verbindung wird in einer Client-Struktur gespeichert und zu der Update-Schleife des Servers hinzugefügt.

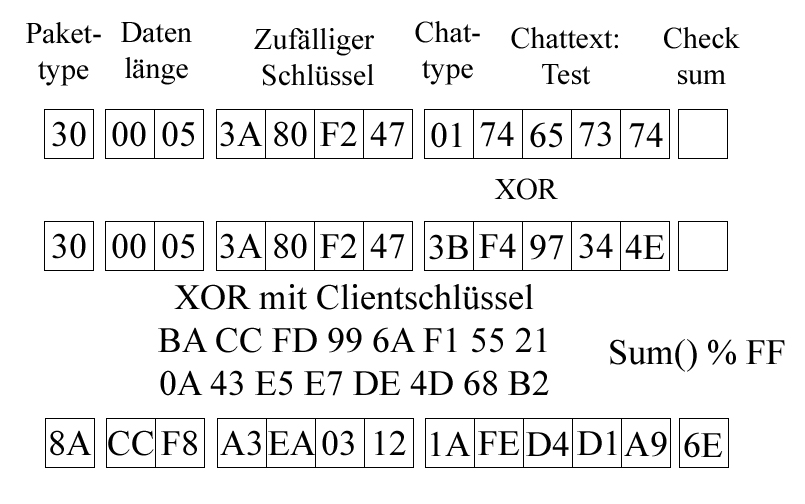
Nun kann der Client das Login-Paket senden. Dieses enthält den eingegebenen Benutzernamen und das Passwort. Der Server überprüft nun in der Datenbank ob das Passwort mit dem Benutzernamen übereinstimmt. Bei erfolgreicher Überprüfung, erstellt der Server einen neuen Spieler und sendet dessen Position, Pokémon und Items an den Client. Anderenfalls wird eine Fehlermeldung übertragen.

Nachdem der Client fertig geladen hat, beginnt der normale Spiel Verlauf.

## 6.1 Verschlüsselung

Alle Pakets werden mit einer doppelten XOR-Operation verschlüsselt. Zuerst wird für das Paket ein 4 Byte langer Schlüssel zufällig generiert. Dieser wird zusammen mit dem Pakettype und der Paketlänge im Header des Pakets verpackt. Der Inhalt des Pakets wird nun mit dem generierten Schlüssel verschlüsselt. Anschließend wird das gesamte Packet mit dem Client-Schlüssel verschlüsselt. Zu aller Letzt wird eine Checksum des Pakets berechnet und als letzes Byte an das Paket angehangen. Diese Checksum berechnet sich aus dem Rest der Division der Summe aller Bytes in dem Packet und 255.

Beispiel eines Chat-Pakets: (alle Zahlen in Hex-Form)



# 7. Map-Management

Die Welt, in der, der Spieler sich bewegen kann, ist nicht eine quadratische Fläche sondern, ist in kleinere quadratische Teile unterteilt. Diese Abschnitte werden untereinander verknüpft, indem man sagt welches Map Element oben, rechts, links und unten angrenzt. Zusätzlich haben alle Maps eine absolute Position, d.h. sie dürfen sich nicht überschneiden. Zur eindeutigen Identifikation sind besitzt jede Map einen Map Index.

Die Zuordnung der Maps zu ihnen Indexen geschieht in der MapIndex.txt. Hier wird der Index einem Ordnernamen zugeordnet in der die Map zu finden ist. In dem Ordner befinden sich das Hintergrundbild, die Wegdaten, die Einstellungen und die Objekte der Map.

Das Hintergrundbild ist ein einfacher Screenshot der organalen Map und wird ganz im Hintergrund gerendert.

Die Wegdaten sind in der \*.atr Datei gespeichert und werden deswegen Atrdata genannt. Siehe dazu *7.1 Wegdaten*.

Die Einstellungen und Objekte werden in den \*.script Dateien gespeichert. Zu den Einstellungen gehören, außer den oben genannten, noch der Type der Map, ob es eine Stadt, Route oder Haus ist und die Anzahl der Objekte auf der Map. Des Weiteren werden die Wilden Pokémon der Map hier eingetragen. Dabei gibt es zwei verschiedene Typen. Die Einen, die in Höhlen oder im Gras zu finden sind und die anderen auf die man im Wasser trifft. Die Objekte sind durchnummeriert und haben zwei Basisinformationen; die lokale Position auf der Map und den Type des Objektes. Je nachdem von welchem Type das Objekt ist, besitzt es unterschiedliche Zusatz Informationen. Bei einem NPC sind dies: der Name, die Textur, das Verhalten, die Quest Funktion, die beim Ansprechen aufgerufen werden soll, der Standard Text falls keine Quest vorhanden ist und die Pokémon, falls es ein Trainer ist. Bei einem Teleporter sind es lediglich die Ziel Map und die Ziel Koordinaten. Bei einem Schild der Text und bei einem Trigger die Quest Funktion die beim Eintreten aufgerufen werden soll. Eine Übersicht über alle Einstellungen ist in *9.4.1 Map Einstellung Beschreibung* zu finden. Alle Daten werden durch den Script-Phraser eingelesen, siehe dazu *7.2 Script-Phraser*.

## 7.1 Wegdaten

Die gesamte Pokémon Welt besteht aus Feldern die 16x16 Pixel groß sind. Die Wegdaten beschreiben von welchen Type ein Feld ist. Es gibt 12 verschieden Typen:

* Weg: Hier kann man ungestört lang laufen
* Wall: Hier kann man nicht lang laufen
* Gras: Hier können Wilde Pokémon auftauchen
* Klippe: Hier kann man nur von einer Richtung drüber laufen (gibt es in alle 4 Richtungen)
* Wasser: Hier muss Surfer eingesetzt werden um weiter zu kommen, außerdem können Wilde Pokémon auftauchen
* Stein: Hier muss Zertrümmerter eingesetzt werden
* Fels: Hier muss Stärke eingesetzt werden
* Pflanze: Hier muss Zerschneider eingesetzt werden
* Wasserfall: Hier muss Kaskade eingesetzt werden

In der atr-Datei ist eine einfaches 2D-Feld aus diesen Typen abgespeichert. Wobei jeder Eintrag ein Feld der Map repräsentiert. Auf diese Weise ist eine Kollisionsabfrage sehr einfach. Man kann einfach gucken ob das nächste Feld begehbar ist bzw. bereits belegt ist. Aus Performance Gründen ist eine Kollisionsabfrage sowohl im Client als auch im Server implementiert.

## 7.2 Script-Phraser

Dieses Module dient zum einlesen von \*.script Dateien. Diese Dateien sind in folgendem Schema aufgebaut:

Kategorie //Im Code Container genannt

{

Integer = 153 //Kommentar

Float =14.8

Vektor = 45,69

Text = Text //Bei einzeiligen Text können “ weggelassen werden

MultilineText = “Das ist ein mehrzeiliger Text

Und das ist die zweite Zeile“

}

Von Prinzip her ist die Datei so aufgebaut wie eine Konfigurationsdatei (\*.ini) von Windows. Es gibt eine Kategorie und darin folgen mehre Items. Der wesentliche Unterschied liegt zum einen darin, dass man in einer ini-Datei keine mehrzeiligen Texte verwenden kann. Zum anderen aber auch, dass man mit dem Modul direkt die Datentypen zurückbekommt.

Wenn eine Datei eingelesen werden soll, wird zuerst solange Zeile für Zeile gelesen bis eine `{` gefunden wurde, also bis ein Container beginnt. Die Zeile zuvor wird zu dem Namen des Containers. Als nächstes werden die Items eingelesen und die Zeile einfach mit dem `=` geteilt. Eine Ausnahme bildet der mehrzeilige Text, dieser wird erkannt indem am Anfang des Wertes ein `“` steht und keins am Ende des Wertes. Bevor allerdings die Zeile analysiert wird, werden zuerst die Kommentare und überflüssige Leerzeichen und Tabs entfernt.

Die Container werden dann in einer Liste von Containern, die jeweils noch eine Liste von Items beinhalten. Da das ganze ursprünglich mit unter Containern gedacht war, besitzt ein Container noch eine Liste von Containern, allerdings habe ich beim weitern programmieren keine Verwendung für Subcontainer gefunden.

# 8. Ergebnis

## 8.1 Übersicht über das Spiel

## 8.2 Selbstkritik

## 8.3 Mögliche Erweiterungen

# 9. Anhang

## 9.1 Quellen

*Pokémon – PokéWiki - Die deutsche Pokémon-Enzyklopädie*. Abgerufen am 12. 02 2013 von PokéWiki: http://www.pokewiki.de/Pokémon

## 9.2 Anleitung zum Spielen

Zuerst muss der Server gestartet werden. Sobald in der Konsole steht, dass dieser bereit ist, kann der Client gestartet werden. Es sind zwei Accounts in der Datenbank registriert. Die Benutzernamen lauten: 'test' und 'test2'. Bei beiden ist das Passwort: 'test'. Nun kann sich mit einem von beiden Eingeloggt werden. Falls ein weiterer Account erstellt werden muss, kann dies per Konsolen-Command geschehen (siehe *9.2.2 Konsolen-Befehle*). Nachdem der Client Fertig geladen hat, befindet man sich in der Spiel-Welt.

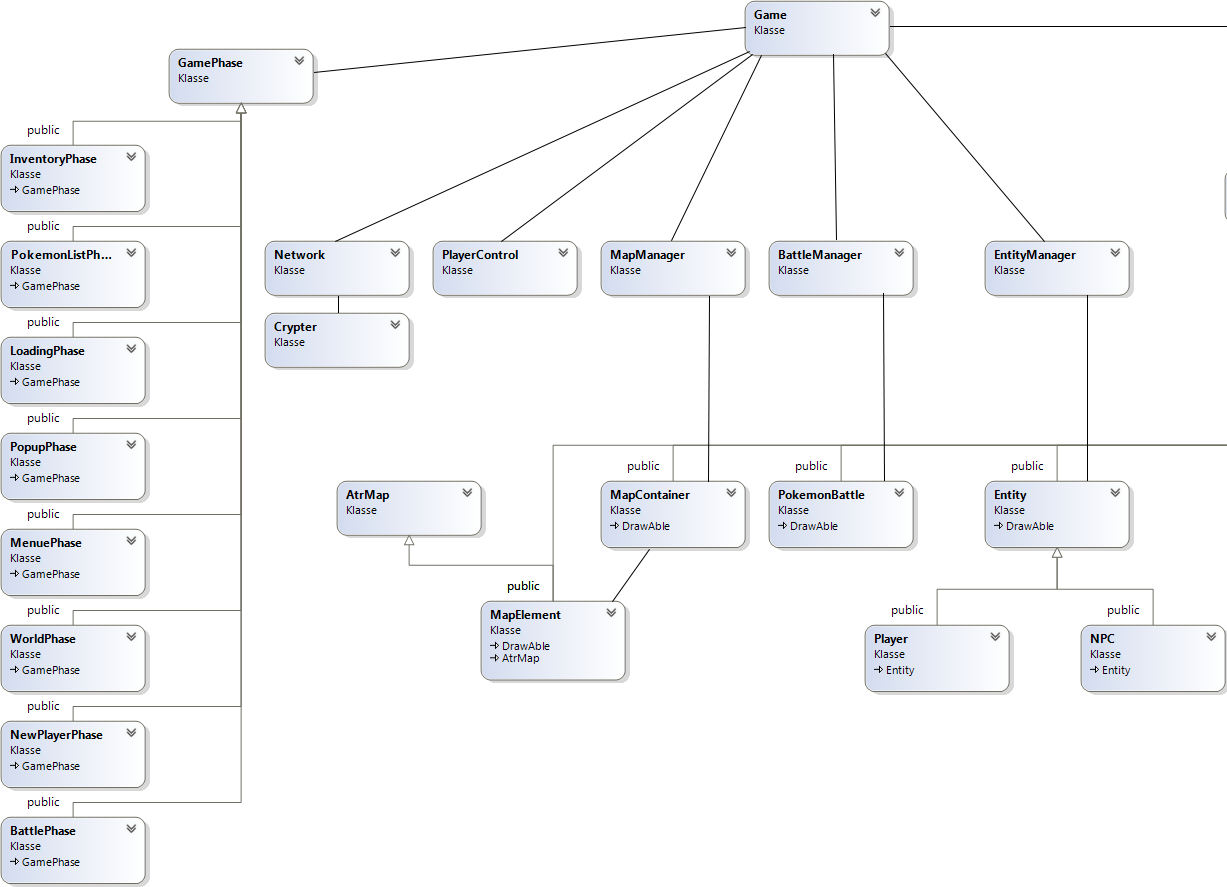
### 9.2.1 Tastaturbelegung

|  |  |  |
| --- | --- | --- |
| Aktion | Taste | Alternativ |
| Hoch-Bewegen | W | Pfeil-Hoch |
| Rechts-Bewegen | D | Pfeil-Rechts |
| Links-Bewegen | A | Pfeil-Links |
| Runter-Bewegen | S | Pfeil-Runter |
| Aktion, Sprechen, Auswählen | Leertaste |  |
| Zurück, Abrechen | ESC |  |
| Dialog schließen | Leertaste | ESC |
| Spiel Menü | ESC |  |
| Beutel | I |  |
| Pokémon | P |  |
| Spielerinfo | C |  |
| Karte | M |  |

### 9.2.2 Konsolenbefehle

## 9.3 Diagramme

### 9.3.1 Client Klassendiagram (Game Abschnitt)



### C:\Users\Niels\Desktop\pwo-2.png9.3.2 Client Klassendiagram (Engine Abschnitt)

### 9.3.3 Server Klassendiagram

### 9.3.4 Datenbank ER-Modell

### 9.3.5 Map-Editor Klassendiagram

## 9.4 Code Ausschnitte

### 9.4.1 Map-Einstellung Beschreibung

MAP //Allgemeine Informationen über die Map

{

name=Homeland //Name der Map

size=20,20 //Größe der Map x,y

position=0,0 //WorldPosition der Map

type=City //Type der Map (City, House, Route, Cave, Forest, Mountain, Underwater, Spezial, Hide)

top=2 //Map am oberen Rand der Map

left=0 //Map am linken Rand der Map

right=0 //Map am rechten Rand der Map

bottom=0 //Map am unteren Rand der Map

underground=0 //Map unter bzw. über der Map z.B. Unterwasser

objectcount=3 //Anzahl der Objekte auf der Map

}

gras\_pokemon //Wilde Pokemon die im hohen Gras auftauchen

{

(1,5,7)=100 //Pokemon (vnum, min-level, max-level) = Chance

}

water\_pokemon //Wilde Pokemon die im Wasser auftauchen

{

(1,5,7)=100 // Pokemon (vnum, min-level, max-level) = Chance

}

OBJECT 1 //Objekt mit der ID 1

{

position=0,0 //Position des Objektes

type=NPC //Type des Objektes

//Zusätzliche Informationen entsprechend des Typs

name=TestNPC //Name des NPCs

texture=Opa001 //Textur des NPCs

behavior=1 //Verhalten des NPCs kann Kombiniert werden

talk=Quest1 //Lua Funktion die ausgeführt wird beim Ansprechen / kann "" sein

text=BlaBla //Standard Text, nur wenn talk="" / kann "" sein

battle=(1,5,1,2,3,4)- //Standard Kampf, nur wenn talk="" / kann "" sein

//Pokemon (vnum, level, attack1-4) mit - getrennt

}

OBJECT 2 //Objekt mit der ID 2

{

position=0,0 //Position des Objektes

type=WARP //Type des Objektes

//Zusätzliche Informationen entsprechend des Typs

destposition=0,0 //Koordinaten des Ziels

destindex=0 //Ziel MapIndex

}

OBJECT 3 //Objekt mit der ID 3

{

position=0,0 //Position des Objektes

type=SIGN //Type des Objektes

//Zusätzliche Informationen entsprechend des Typs

text="Ein Test Schild" //Text auf dem Schild

}

OBJECT 4 //Objekt mit der ID 4

{

position=0,0 //Position des Objektes

type=TRIGGER //Type des Objektes

//Zusätzliche Informationen entsprechend des Typs

enter=Quest1 //Lua Funktion die ausgeführt wird beim Eintreten

}

1. Multiplayer – (eng.) Mehrspieler [↑](#footnote-ref-1)
2. Story – (eng.) Geschichte hier: die Geschichte des Protagonisten im Spiel [↑](#footnote-ref-2)
3. Item – (eng.) Gegenstand [↑](#footnote-ref-3)
4. Quest – (eng.) Suche - hier: Aufgaben des Spielers [↑](#footnote-ref-4)
5. Wegdatan – siehe 7.1 Wegdaten [↑](#footnote-ref-5)
6. UI – User Interface – (eng.) Benutzeroberfläche [↑](#footnote-ref-6)
7. Map - (eng.) Karte [↑](#footnote-ref-7)
8. Singleton-Klasse – Eine Klasse die nur eine statische Instanz besitzt [↑](#footnote-ref-8)
9. Framerate - Frames pro Sekunde [↑](#footnote-ref-9)
10. Backbuffer - (eng.) Hintergrundspeicher [↑](#footnote-ref-10)
11. pseudo 3D-Effeck - Überlappung von 2D-Grafiken die einen Tiefeneffekt hervorrufen [↑](#footnote-ref-11)
12. Control - (eng.) Steuerelement [↑](#footnote-ref-12)